***TRAINING AN IMAGE CLASSIFIER***

Task 1 Results Reproduction snapshot:

A screenshot of a computer

Description automatically generated

Task 2 Failure Cases:

A yellow fish with black text

Description automatically generatedA close up of a horse's head

Description automatically generatedA white and brown dog

Description automatically generatedA white bird in the sky

Description automatically generatedA picture of a plane

Description automatically generated

Task 3 Hyperparameter Tuning:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Epochs | Batch Size | Learning Rate | Momentum | Test Accuracy |
| 2 | 4 | 0.001 | 0.9 | 54% |
| 4 | 4 | 0.001 | 0.9 | 58% |
| 2 | 8 | 0.001 | 0.9 | 51% |
| 2 | 4 | 0.0005 | 0.9 | 51% |
| 2 | 4 | 0.001 | 0.45 | 43% |
| 4 | 8 | 0.0005 | 0.45 | 38% |
| ***4*** | ***4*** | ***0.001*** | ***0.9*** | ***58%*** |

Task 4 Dataset augmentation:

So, what I did was initially ran the model to replicate the accuracy from the tutorial. Then ran it again using the transformed dataset to see what happened. But I also did one where it was only ran using the augmented set.

1. There where 3 types of augmentations applied: Horizontal flips, Rotations, Translations. They were all random. The flips were applied randomly with a fixed probability of 50%. The rotations were applied with a random degree between -20 and 20 degrees. The translations can be up to 10% of the image’s width to the left/right or up/down.
2. The performance did not improve when running the model on the augmented set after running it on regular set first. Performance dropped from 54% accuracy to 52% accuracy. Running the model only on the augmented set gives a performance of 46%. However, this was with the hyperparameters tuned to what the tutorial had them as. When I increased the epoch from 2 to 8, and run the model once on the augmented dataset, I get an accuracy of 56%. So not much of an increase from the original, but an increase, nonetheless.
3. Data augmentation helped improve accuracy b/c the transformations diversified the training examples. The model now has more orientations/perspectives to learn from which will help with future predictions when testing.
4. I think brightness/dimness and scaling transformations can be of good use. The scaling transformations would help b/c images are not always taken from an ideal distance away, it can be taken from very up close or very far away. Similarly, the lighting for pictures isn’t always ideal, so presenting the model with different variations of these transformations will deepen it’s learning and increase accuracy.

Code to augment:

transform\_augmented = transforms.Compose([

transforms.RandomHorizontalFlip(), # Flip the image randomly with a given probability

transforms.RandomRotation(20), # Random rotation between -20 and 20 degrees

transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)), # Random translation

transforms.ToTensor(), # Convert images to tensors

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) # Normalize images

])

Output of augmentations:

A collage of images of an airplane and a lizard

Description automatically generatedA collage of images of a plane and a bird

Description automatically generated

Task 5 (Extra Credit):

1. The classifier’s prediction would be completely wrong. I ran some code and the ground truth output was this, then when I ran the code to get the prediction, it read “dog dog bird dog”  
   Code:  
     
   #MNIST STUFF

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

transform\_mnist = transforms.Compose([

transforms.Grayscale(3), # Convert MNIST images to 3-channel

transforms.Resize((32, 32)),

transforms.ToTensor()

])

mnist\_test = datasets.MNIST(root='./data', train=False, download=True, transform=transform\_mnist)

mnist\_loader = DataLoader(mnist\_test, batch\_size=64, shuffle=False)

dataiter = iter(mnist\_loader)

images, labels = next(dataiter)

# print images

imshow(torchvision.utils.make\_grid(images))

print('GroundTruth: ', ' '.join(f'{classes[labels[j]]:5s}' for j in range(4)))  
  
Output:  
A screenshot of a computer program

Description automatically generatedA grey square with white numbers

Description automatically generated

1. Well for seen classes the model would give high confidence because image features would resemble features in the training data. Whereas for unseen classes the confidence would vary unpredictably because the image features may have patterns the model thinks it recognizes where other patterns it doesn’t.
2. Well, some kind of unseen class detection could be implemented using some SoftMax threshold that will flag or refuse the image, almost like how a bouncer checks IDs before letting people into a club. Similarly, the CNN would check against the threshold before giving any predictions.

***EVALUATION OF OBJECT DETECTION MODELS***

**Task 1:** *Load Object Detector Models*

*Code:*

# Download a pretrained model

model0 = torchvision.models.detection.fasterrcnn\_resnet50\_fpn(pretrained=True)

# maskrcnn\_resnet50\_fpn

model1 = torchvision.models.detection.maskrcnn\_resnet50\_fpn(pretrained=True)

# retinanet\_resnet50\_fpn

model2 = torchvision.models.detection.retinanet\_resnet50\_fpn(pretrained=True)

# ssdlite320\_mobilenet\_v3\_large

model3 = torchvision.models.detection.ssdlite320\_mobilenet\_v3\_large(pretrained=True)

# Inference

model0.eval()

model1.eval()

model2.eval()

model3.eval()

**Task 2:** *Object Detection Pipeline*

*Code:*

def get\_detection(img, model, threshold=0.5):

pred = model([img]) # Pass the image to the model

# pred is a list and each element of that list is a dictionary with keys: "labels", "scores", and "boxes"

pred\_classes = [COCO\_INSTANCE\_CATEGORY\_NAMES[i] for i in list(pred[0]['labels'].numpy())] # Get the Prediction Classes

# !!!! Complete the following (edited the next 3 lines)

pred\_boxes = pred[0]['boxes'].detach().numpy() # Get the Prediction Boxes

pred\_scores = pred[0]['scores'].detach().numpy() # Get the Prediction Scores

high\_score\_indices = [i for i, score in enumerate(pred\_scores) if score > threshold]

if high\_score\_indices:

last\_index = high\_score\_indices[-1] + 1

pred\_boxes = pred\_boxes[:last\_index]

pred\_classes = pred\_classes[:last\_index]

else:

pred\_boxes = np.array([]) # In case no scores are above threshold

pred\_classes = []

return pred\_boxes, pred\_classes

**Task 3:** *Display Detections*

*Code:*

def show\_detections(img\_path, model, threshold=0.5):

img = Image.open(img\_path) # Load the image

img = transform(img) # Apply the transform to the image

boxes, pred\_cls = get\_detection(img, model, threshold) # Get predictions

img = cv2.imread(img\_path) # Read image with cv2

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) # Convert to RGB

# !!!! Complete the following

# write code to display the image, overlay the bounding boxes and predicted classes on top of the image

fig, ax = plt.subplots(figsize=(12, 8)) # Set up the figure and axes

ax.imshow(img)

# !!!!

for box, cls in zip(boxes, pred\_cls):

x, y, xmax, ymax = box

width = xmax - x

height = ymax - y

# Draw a rectangle around each box

rect = patches.Rectangle((x, y), xmax - x, ymax - y, linewidth=2, edgecolor='red', facecolor='none')

ax.add\_patch(rect)

# Add label

ax.text(x, y, cls, verticalalignment='top', color='white', fontsize=10, backgroundcolor='red')

plt.show()

!wget https://www.tejasgokhale.com/images/vehicle.png -O vehicle.jpg

# code to use show\_detections to display results for `model0,model1,model2,model3`

# !!!! Complete this

img\_path = 'vehicle.jpg'

show\_detections(img\_path, model0, threshold=0.5)

show\_detections(img\_path, model1, threshold=0.5)

show\_detections(img\_path, model2, threshold=0.5)

show\_detections(img\_path, model3, threshold=0.5)

*Output:*

Model0 || Model1:

A group of people standing in a road

Description automatically generatedA group of firefighters standing next to a car

Description automatically generated

Model2 || Model3:

A group of people standing in a road with a car in the background

Description automatically generatedA group of firefighters standing next to a car

Description automatically generated

**Task 4:** *Evaluation using MS-COCO dataset.*

Calculate meanIOU over entire dataset of 100 images and report meanIOU for all 4 models in a table:

Code:

coco\_annotations\_file="/Users/fneba/Desktop/691\_Computer\_Vision/hw4/coco\_ann2017/annotations/instances\_val2017.json"

coco\_images\_dir="/Users/fneba/Desktop/691\_Computer\_Vision/hw4/coco\_val2017/val2017"

coco= COCOParser(coco\_annotations\_file, coco\_images\_dir)

img\_ids = coco.get\_imgIds()

img\_ids = img\_ids[:100]

# For mean IoU calculation

mean\_ious0 = [] # added for model0

mean\_ious1 = [] # added for model1

mean\_ious2 = [] # added for model2

mean\_ious3 = [] # added for model3

for i, im in enumerate(img\_ids):

image = Image.open(f"{coco\_images\_dir}/{str(im).zfill(12)}.jpg")

image = transform(image)

pred\_boxes0, pred\_class0 = get\_detection(image, model0)

pred\_boxes1, pred\_class1 = get\_detection(image, model1)

pred\_boxes2, pred\_class2 = get\_detection(image, model2)

pred\_boxes3, pred\_class3 = get\_detection(image, model3)

ann\_ids = coco.get\_annIds(im)

annotations = coco.load\_anns(ann\_ids)

# Calculate IoU for each model

mean\_ious0.append(eval\_iou(pred\_boxes0, pred\_class0, annotations))

mean\_ious1.append(eval\_iou(pred\_boxes1, pred\_class1, annotations))

mean\_ious2.append(eval\_iou(pred\_boxes2, pred\_class2, annotations))

mean\_ious3.append(eval\_iou(pred\_boxes3, pred\_class3, annotations))

print(f"Mean IoU for model0 over 100 images: {np.mean(mean\_ious0)}")

print(f"Mean IoU for model1 over 100 images: {np.mean(mean\_ious1)}")

print(f"Mean IoU for model2 over 100 images: {np.mean(mean\_ious2)}")

print(f"Mean IoU for model3 over 100 images: {np.mean(mean\_ious3)}")

Output:

|  |  |
| --- | --- |
| Model | meanIOU |
| model0 | 0.5766270008358215 |
| model1 | 0.5795787956734314 |
| model2 | 0.6217368327239078 |
| model3 | 0.6054661987556592 |

Similarly, report the precision and recall of each model:

Code:

def calculate\_matches(pred\_boxes, pred\_classes, annotations, iou\_threshold=0.5):

TP = 0

FP = 0

FN = 0

matched\_gt\_indices = set() # Keep track of matched ground truth indices

# Check each prediction for potential matches

for pred\_box, pred\_class in zip(pred\_boxes, pred\_classes):

found\_match = False

for idx, ann in enumerate(annotations):

gt\_bbox = [ann['bbox'][0], ann['bbox'][1], ann['bbox'][0] + ann['bbox'][2], ann['bbox'][1] + ann['bbox'][3]]

gt\_class\_id = ann["category\_id"]

pred\_class\_id = COCO\_INSTANCE\_CATEGORY\_NAMES.index(pred\_class) if pred\_class in COCO\_INSTANCE\_CATEGORY\_NAMES else -1

if pred\_class\_id == gt\_class\_id and iou(pred\_box, gt\_bbox) >= iou\_threshold:

if idx not in matched\_gt\_indices:

matched\_gt\_indices.add(idx)

found\_match = True

TP += 1

break

if not found\_match:

FP += 1

# Compute FN as ground truths that were not matched

FN = len(annotations) - len(matched\_gt\_indices)

return TP, FP, FN

models = [model0, model1, model2, model3]

TPs = [0, 0, 0, 0] # TP for model0, model1, model2, model3

FPs = [0, 0, 0, 0] # FP for model0, model1, model2, model3

FNs = [0, 0, 0, 0] # FN for model0, model1, model2, model3

for im in img\_ids:

image\_path = f"{coco\_images\_dir}/{str(im).zfill(12)}.jpg"

image = Image.open(image\_path).convert('RGB')

image = transform(image)

ann\_ids = coco.get\_annIds(im)

annotations = coco.load\_anns(ann\_ids)

for idx, model in enumerate(models):

pred\_boxes, pred\_classes = get\_detection(image, model)

tp, fp, fn = calculate\_matches(pred\_boxes, pred\_classes, annotations)

TPs[idx] += tp

FPs[idx] += fp

FNs[idx] += fn

for idx, model in enumerate(models):

precision = TPs[idx] / (TPs[idx] + FPs[idx]) if TPs[idx] + FPs[idx] > 0 else 0

recall = TPs[idx] / (TPs[idx] + FNs[idx]) if TPs[idx] + FNs[idx] > 0 else 0

print(f"Precision for model{idx}: {precision:.2f}")

print(f"Recall for model{idx}: {recall:.2f}")

Output:

|  |  |  |
| --- | --- | --- |
| Model | Precision | Recall |
| model0 | 0.50 | 0.71 |
| model1 | 0.52 | 0.74 |
| model2 | 0.76 | 0.53 |
| model3 | 0.80 | 0.26 |

***REQUIRED FOR 691: GUEST LECTURES***

**Yu Zeng:**

1. I wasn’t really taking notes when the lecturer was speaking, but the lecturer’s talk was creating and improving a model that can generate images from text. These images should somewhat mirror a real picture of what was asked. For example, Dr. Yu Zeng has examples of cartoon characters being generated from their model, the “accuracy” of the model would be how close that generated image is to the original picture of the cartoon character.
2. I asked a relatively simple question about her upcoming research. Just what exactly was the purpose of trying to recreate image of diseases/viruses, when you have REAL images of them? She said it would be to try and generate images, en masse, that can depict illnesses so that models that are used to detect certain illnesses have an image set to train on so that their performance gets better. Model for image data set generation so that the models that do disease detection can get better.
3. My favorite portion was honestly the part about her further research. Because as it stood, I didn’t really see the utility in being able to generate pictures that looked like other pictures. But then when the lecturer brought up how her research could be useful in the health industry that’s when I started to realize the applications. For example, another industry it could potentially help in is animation/entertainment. If the model is trained well enough to replicate characters, it could help with drawing the different still frames for things like anime which may help quicken the process of animating them into episodes (just a theory).
4. Y. Zeng, Z. Lin, J. Zhang, Q. Liu, J. Collomosse, J. Kuen, V. Patel, “Scenecomposer: Any-level semantic image synthesis,” CVPR, 2023 (Hightlight, top 2.5% )  
   Y. Zeng, V. M. Patel, H. Wang, X. Huang, T. Wang, M. Liu, Y. Balaji, “Jedi: Joint-image diffusion models for finetuning-free personalized text-to-image generation,” CVPR, 2024.