

Project 1 - Cat & Dog Classification

Henrik Daniel Christensen [hench13@student.sdu.dk]
Frode Engtoft Johansen [fjoha21@student.sdu.dk]

DM873: Deep Learning
University of Southern Denmark, SDU
Department of Mathematics and Computer Science

1 Introduction

The objective of this project is to develop a deep learning model capable of distinguishing between images of cats and dogs. The task involves training a neural network using a dataset containing 3,600 images, equally divided between the two categories.

Note that the code and notebook is available in the Appendix.

2 Explorative Analysis

The dataset contains a mix of different breeds of cats and dogs. The images are very diverse in terms of size, orientation, lighting, focus, and resolution. Also, the images are taken from different angles and distances. The images are mostly around 200-500 pixels width/height and are all in color, meaning they have 3 color channels. In Figure 1, examples of cats and dogs from the dataset are shown.

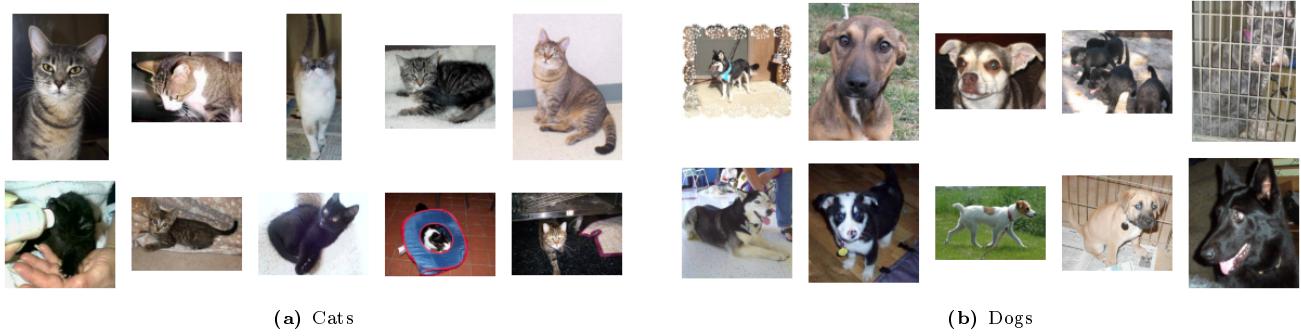


Fig. 1. Sample images of cats and dogs from the dataset.

From the images, clearly, cats and dogs have different features that can be used to distinguish between them. For example, cats have generally shorter faces often with triangular ears and pointed noses, while dogs generally have longer snouts and have more different ear shapes. Moreover, the eyes of cats are generally more sharp-shaped, while dogs have rounder eyes. The goal is therefore to create a model that can learn some of these features.

3 Base Model

First, we set out to develop a base model. However, first, we had to decide which image size to use. We chose an image size of 224x224, as this size seems to capture enough details. Also, many of the pretrained models uses the same image size, making our model more comparable to these. Resizing the images has the effect of making training of the model significantly faster.

Next, we had to decide how many layers our model should have. As the task is relatively simple, we decided to use 4 convolutional layers and 3 fully connected layers, including the binary output layer. Since the input images are in color and 224x224 pixels, inputting that image directly into the fully connected layers would result in a $224 \cdot 224 \cdot 3 = 150528$ input size into the model. We would like to reduce this further, so the fully connected layers does not get an input of 150528. The convolution layers extract features, which makes it possible for the neural network to recognise a specific feature no matter where it is in the image, instead of having to learn that feature for every possible position in an image.

We use cross-entropy loss as our loss function since this loss function is a generally good and popular for classification tasks. We use Adam as the optimizer since it combines the strength of several other optimizers.

#TODO: Mangler at beskrive activation function, kernel size og maxpooling

The base model is shown in Table 1.

	Output kernels/features	Kernel	MaxPooling	Activation
Conv2D w/	32	3x3	2x2	ReLU
Conv2D w/	64	3x3	2x2	ReLU
Conv2D w/	128	3x3	2x2	ReLU
Conv2D w/	256	3x3	2x2	ReLU
Linear w/	256	-	-	ReLU
Linear w/	128	-	-	ReLU
Linear w/	2	-	-	-

Table 1. Base Model.

We chose to train the model in 25 epochs with a learning rate of 0.001 as this generally is a good choice. A learning rate that is too small could result in a very slow learning process that might get stuck, and a too big learning rate could result in learning a suboptimal set of weights or an unstable learning process. Results of the base model after 25 epochs are shown in Figure 2.

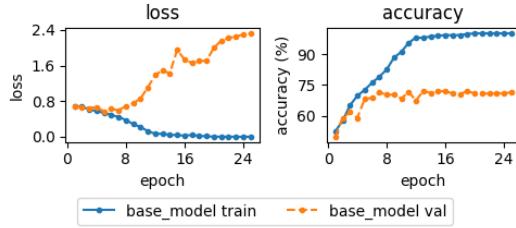


Fig. 2. Base Model Results.

Clearly, the base model is overfitting, therefore, we need to add some regularization to the model.

4 Regularization

4.1 Data Augmentation

One way, especially for small datasets as in this case, to reduce overfitting is to use data augmentation, enabling the model to learn from more data.

We chose to augment the data in a few different ways since we want to generalize the data so the important features remain. Using augmentation ensures that it will get better accuracy when faced with new images. In other words, it reduces overfitting.

The images are of varying quality and taken in different lighting, and this is also what we can expect from the final test set. For this reason we change the brightness, saturation, and contrast. Furthermore, the pictures are taken from different angles so we rotate the pictures slightly and flip them horizontally. Generally, images of animals are not vertically flipped, so we chose not to do this, but of course, some cases may exist.

The pictures are also cropped differently to get finer details from the images, as well as the images becomes scale invariant.

Even though some of the pictures are more blurry than others, we chose to not use blurring as data augmentation, as it effectively just reduces the data of the image without reducing input to the model.

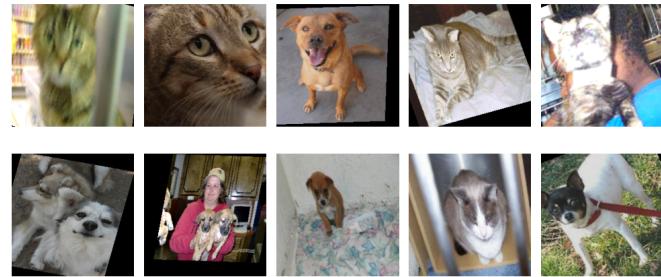
We shear and translate the pictures to emulate pictures taken from different angles of the animal.

Since the training dataset is quite small, we wanted to create more data by reusing the same images but with random augmentations, but in the end we found out that our results did not improve from this.

Many different data augmentation techniques exists, to find the best data augmentation techniques for this task, one image was used to test different techniques.

The techniques used for this task as well as augmented sample images by these techniques are shown in Figure 3.

Data Augmentation	Parameters
RandomHorizontalFlip	50%
ColorJitter	brightness=0.2 contrast=0.2 saturation=0.2 hue=0
RandomAffine	degrees=25 translate=(0.1, 0.1) scale=(0.7, 1.3) shear=(-10, 10)



(a) Data Augmentation Techniques.

(b) Sample images after augmentation.

Fig. 3. Data augmentation techniques.

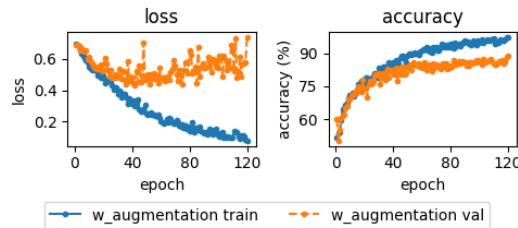


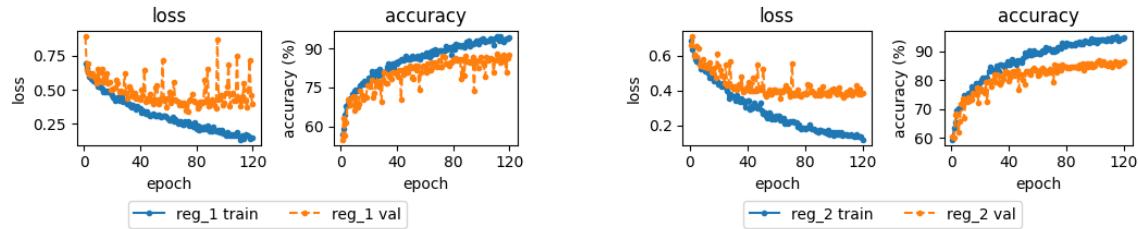
Fig. 4. Sample images after augmentation.

4.2 More Regularization Techniques

1: - 93- 84

2: - 93- 85

- loss mere stabil på 2. Den er med weight decay og halvering af learning rate hver 25 epoker. - dropout layer

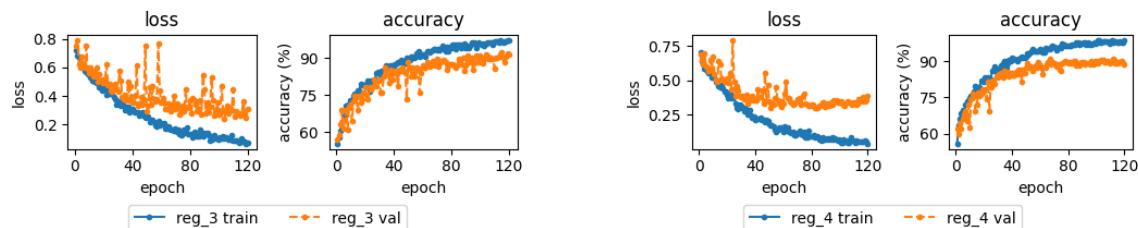


(a) Regularized model 1.

(b) Regularized model 2.

Fig. 5. Results using regularization.

4.3 Adding one more convolutional layer



(a) Regularized model 3.

(b) Regularized model 4.

Fig. 6. Results using one more convolutional layer.

#TODO: Tilføj feature maps

5 Prediction

For prediction on the test set, we choose the model 'Regularized model 4' as it achieves the highest validation accuracy. The model is applied to the test set, with its accuracy result summarized in Table 2.

Test Accuracy using Reg4 model

91.5%

Table 2. Test Accuracy using Reg4 Model.

Some of the misclassified and correctly classified images are given in Figure 7.



Fig. 7. Misclassified and correctly classified images.

From the misclassified images, we see that some more data augmentation could be applied to improve the model's performance. For example, the cat that rotated its head by almost 90 degrees. Moreover, the model may be improved by adding more crop-scale augmentation.

6 Pretrained Model (AlexNet)

To see how good our model performs compared to a well-known model, we used the pretrained model 'AlexNet'. This model is pretrained on the ImageNet dataset, which contains 1.2 million images and 1000 classes, also including cats and dogs. However, to use the pretrained model for this binary classification task, the output layer is modified. The results of the pretrained model after 20 epochs are shown in Figure 8a. The test accuracy is summarized in Table 8b.

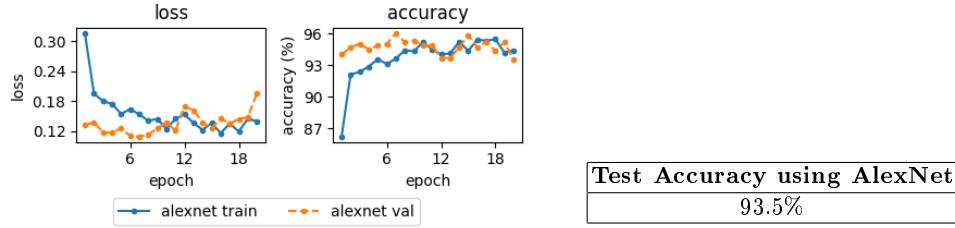


Fig. 8. AlexNet results.

As expected the pretrained model performs slightly better than our model. However, AlexNet also is trained on a much larger dataset and has a lot more parameters (~ 57 million) compared to our model (~ 21 million).

7 Conclusion

We ended up with a model that has ?? results. The model we used was ?? with ?? layers having XX parameters. This result is pretty good, but it could be improved. It is worth noticing that some of the pictures are very hard to classify, like dog.1335 and dog.1395.

- compare results to pretrained model - why is the pretrained model better?

The model could have been improved in various ways. First of all more training data would very likely improve the model a lot. In some of the pictures its very hard to see whether its a cat or a dog and its doubtful that the model would learn anything from the picture, so removing those pictures from the dataset would potentially help the model learn more meaningful features and also train faster. We could have run epochs until the model began overfitting and then stopped at that point. If time was not a factor this is probably what we would have done. We could have also experimented more with the different parameters like learning rate and weight decay. We could have tried even more data augmentations to see what works best. We only rotate the pictures up to 25 degrees, which means that the model as an example would have a hard time recognising pictures of animals upside down.

7.1 Individual Contributions

	Henrik Daniel Christensen	Frode Engtoft Johansen
Code	<ul style="list-style-type: none"> - Base model - Data Augmentation - Regularization - Pre-trained 	
Report	<ul style="list-style-type: none"> - Introduction - Explorative Analysis - Base Model - Regularization - Prediction - Pretrained Model - Conclusion 	

Table 3. Individual contributions.

A Code

Python Files

File: default_config.py

```

image_size = 224
batch_size = 64
num_workers = 8
pin_memory = True
normalize_mean = [0.485, 0.456, 0.406]
normalize_std = [0.229, 0.224, 0.225]
data_dir = "data"
train_dir = f"{data_dir}/train"
val_dir = f"{data_dir}/validation"
test_dir = f"{data_dir}/test"
label_map = {0: "cat", 1: "dog"}

default_transform_config = {
    "hflip": True,
    "brightness_jitter": 0.2,
    "contrast_jitter": 0.2,
    "saturation_jitter": 0.2,
    "hue_jitter": 0.0,
    "rotation": 25, # rotation is in degrees
    "crop_scale": 0.3, # crop_scale is like zooming in. 0.3 means zoom in by 30%
    "translate": 0.1, # translate is like shifting the image. 0.1 means shift by 10%
    "shear": 10, # shear is like skewing the image. 10 means shear by 10 degrees
}

no_transform_config = {
    "hflip": False,
    "brightness_jitter": 0,
    "contrast_jitter": 0,
    "saturation_jitter": 0,
    "hue_jitter": 0,
    "rotation": 0,
    "crop_scale": 1,
    "translate": 0,
    "shear": 0,
}

default_train_config = {
    "optimizer_type": "Adam",
    "learning_rate": 0.001,
    "weight_decay": 0,
    "momentum": None,
    "reg_type": "None",
    "reg_lambda": 0.001,
    "step_size": None,
    "gamma": None,
}

default_net_config = {
    "in_channels": 3,
    "num_classes": 2,
    "cv_layers": [
        {
            "out_channels": 16,
            "kernel_size": 3,
            "stride": 1,
            "padding": 1,
            "batch_norm": False,
            "max_pool": 0,
            "max_pool_stride": 1,
        },
        {
            "out_channels": 32,
            "kernel_size": 3,
            "stride": 1,
            "padding": 1,
            "batch_norm": False,
            "max_pool": 0,
            "max_pool_stride": 1,
        },
    ],
    "fc_layers": [{"out_features": 64, "batch_norm": False, "dropout_rate": 0}],
}

```

```
default_config = {
```

```
    "label": "Default Experiment",
    "n_epochs": 120,
    "transform_config": default_transform_config,
    "train_config": default_train_config,
    "net_config": default_net_config,
```

```
}
```

File: loaders.py

```
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from default_config import (
    image_size,
    batch_size,
    num_workers,
    pin_memory,
    normalize_mean,
    normalize_std,
    default_transform_config,
    train_dir,
)

def get_train_loader(transform_config=default_transform_config):
    hflip = transform_config["hflip"]

    brightness_jitter = transform_config["brightness_jitter"]
    contrast_jitter = transform_config["contrast_jitter"]
    saturation_jitter = transform_config["saturation_jitter"]
    hue_jitter = transform_config["hue_jitter"]

    rotation = transform_config["rotation"]
    crop_scale = transform_config["crop_scale"]
    translate = transform_config["translate"]
    shear = transform_config["shear"]

    transform_list = [transforms.Resize((image_size, image_size))] # always resizing

    # Add transformations
    if hflip:
        transform_list.append(transforms.RandomHorizontalFlip(p=0.5))

    color_jitter_params = {}
    if brightness_jitter > 0:
        color_jitter_params["brightness"] = brightness_jitter
    if contrast_jitter > 0:
        color_jitter_params["contrast"] = contrast_jitter
    if saturation_jitter > 0:
        color_jitter_params["saturation"] = saturation_jitter
    if hue_jitter > 0:
        color_jitter_params["hue"] = hue_jitter
    if brightness_jitter > 0 or contrast_jitter > 0 or saturation_jitter > 0 or hue_jitter > 0:
        transform_list.append(transforms.ColorJitter(**color_jitter_params))

    affine_params = {}
    if rotation > 0:
        affine_params["degrees"] = rotation
    if crop_scale < 1 and rotation > 0:
        affine_params["scale"] = (1 - crop_scale, 1 + crop_scale)
    if translate > 0 and rotation > 0:
        affine_params["translate"] = (translate, translate)
    if shear > 0 and rotation > 0:
        affine_params["shear"] = (-shear, shear)
    if rotation > 0: # rotation must be set for RandomAffine to work
        transform_list.append(transforms.RandomAffine(**affine_params))

    # Convert to tensor
    transform_list.append(transforms.ToTensor())

    # Normalize
    transform_list.append(transforms.Normalize(mean=normalize_mean, std=normalize_std))

    # Compose all transformations into a single pipeline
    train_transform = transforms.Compose(transform_list)
```

```

# Load data
train_data = datasets.ImageFolder(train_dir, transform=train_transform)

# Create data loader
train_loader = DataLoader(
    train_data,
    batch_size=batch_size,
    shuffle=True,
    num_workers=num_workers,
    pin_memory=pin_memory,
)
return train_loader

def get_test_transform():
    return transforms.Compose(
        [
            transforms.Resize((image_size, image_size)),
            transforms.ToTensor(),
            transforms.Normalize(mean=normalize_mean, std=normalize_std),
        ]
    )

def get_test_loader(dir):
    test_transform = get_test_transform()

    test_data = datasets.ImageFolder(dir, transform=test_transform)

    test_loader = DataLoader(
        test_data,
        batch_size=batch_size,
        shuffle=False,
        num_workers=num_workers,
        pin_memory=pin_memory,
    )
    return test_loader

```

File: convolutionalNetwork.py

```

from default_config import default_net_config, image_size
import torch
import torch.nn as nn

class ConvolutionalNetwork(nn.Module):
    def __init__(self, net_config=default_net_config):
        super(ConvolutionalNetwork, self).__init__()

        in_channels = net_config["in_channels"]
        num_classes = net_config["num_classes"]

        layers = []
        self.relu = nn.ReLU()

        tmp_channels = in_channels
        for config in net_config["cv_layers"]:
            out_channels = config["out_channels"]
            kernel_size = config.get("kernel_size", 3)
            stride = config.get("stride", 1)
            padding = config.get("padding", 0)
            batch_norm = config.get("batch_norm", False)
            max_pool = config.get("max_pool", 0)

            # Convolutional layer
            # - Output channels are the number of filters in the convolutional layer
            # - Kernel size is the size of the filter
            # - Stride is the step size for the filter
            # - Padding is the number of pixels to add around the input image
            layers.append(
                nn.Conv2d(
                    tmp_channels,
                    out_channels,
                    kernel_size=kernel_size,
                    stride=stride,

```

```

        padding=padding,
    )
)

# Batch Normalization
# - Batch normalization is used to normalize the input layer by adjusting and scaling the activations.
#   It is used to make the model faster and more stable.
if batch_norm:
    layers.append(nn.BatchNorm2d(num_features=out_channels))

# ReLU activation
# - An activation function is used to introduce non-linearity to the output of a neuron
layers.append(self.relu)

# Maxpooling layer
# - Max pooling is a downsampling operation that reduces the dimensionality of the input
# - Kernel size is the size of the filter
# - Stride is the step size for the filter
if max_pool > 1:
    max_pool_stride = config.get("max_pool_stride", 1)
    layers.append(nn.MaxPool2d(kernel_size=max_pool, stride=max_pool_stride))

# Update input channels for next iteration
tmp_channels = out_channels

# Sequential container for convolutional layers
self.cv_layers = nn.Sequential(*layers)

# Calculate input size of tensor after convolutional layers
with torch.no_grad():
    sample_input = torch.randn(
        1, in_channels, image_size, image_size
    ) # create a random sample input tensor
    conv_output = self.cv_layers(sample_input)
    fc_input_features = (
        conv_output.numel()
    ) # calculate the number of elements in the tensor

# Fully connected layers
fc_layers_list = []
for config in net_config["fc_layers"]:
    out_features = config["out_features"]
    dropout_rate = config.get("dropout_rate", 0)
    batch_norm = config.get("batch_norm", False)

    # Fully connected layer
    # - Input features are the number of neurons in previous layer
    # - Output features are the number of neurons to create in current layer
    fc_layers_list.append(
        nn.Linear(in_features=fc_input_features, out_features=out_features)
    )

    # Batch Normalization
    # - Batch normalization is used to normalize the input layer by adjusting and scaling the activations.
    #   It is used to make the model faster and more stable.
    if batch_norm:
        fc_layers_list.append(nn.BatchNorm1d(out_features))

    # ReLU activation
    # - An activation function is used to introduce non-linearity to the output of a neuron
    fc_layers_list.append(self.relu)

    # Dropout
    # - Dropout is a regularization technique to prevent overfitting by randomly setting some output features to 0:
    if dropout_rate > 0:
        fc_layers_list.append(nn.Dropout(p=dropout_rate))

    # Update input features for next iteration
    fc_input_features = out_features

# Output layer (no dropout or activation)
fc_layers_list.append(nn.Linear(fc_input_features, num_classes))

# Sequential container for fully connected layers
self.fc_layers = nn.Sequential(*fc_layers_list)

def forward(self, x):
    x = self.cv_layers(x) # pass through convolutional layers

```

```

x = x.flatten(
    start_dim=1, end_dim=-1
) # flatten tensor from convolutional layers for the linear fully connected layers
x = self.fc_layers(x) # pass through fully connected layers
return x

```

File: train_model.py

```

from default_config import default_config, batch_size, image_size, val_dir
from loaders import get_train_loader, get_test_loader
import time
import torch
import torch.nn as nn
import torch.optim as optim
from torchinfo import summary

def train_model(model, device, config=default_config):
    label = config["label"]
    n_epochs = config["n_epochs"]

    train_config = config["train_config"]
    transform_config = config["transform_config"]

    model = model.to(device) # move model to device

    criterion = nn.CrossEntropyLoss() # loss function

    optimizer_type = train_config["optimizer_type"]
    learning_rate = train_config["learning_rate"]
    weight_decay = train_config.get("weight_decay", 0.0)
    momentum = train_config.get("momentum", 0.0)
    reg_type = train_config["reg_type"]
    reg_lambda = train_config["reg_lambda"]
    step_size = train_config["step_size"]
    gamma = train_config["gamma"]

    # Get transformations
    train_loader = get_train_loader(transform_config)
    val_loader = get_test_loader(val_dir)

    # Select optimizer
    if optimizer_type == "SGD":
        optimizer = optim.SGD(
            model.parameters(), lr=learning_rate, momentum=momentum, weight_decay=weight_decay
        )
    elif optimizer_type == "Adam":
        # If weight decay is specified, apply AdamW instead
        if weight_decay > 0:
            optimizer = optim.AdamW(model.parameters(), lr=learning_rate, weight_decay=weight_decay)
        else:
            optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    # Scheduler
    if step_size is not None and gamma is not None:
        scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=step_size, gamma=gamma)

    train_losses = []
    train_accuracies = []
    val_losses = []
    val_accuracies = []

    print(f"\nExperiment: {label}")

    # Track total training time
    total_start_time = time.time()

    for epoch in range(n_epochs):

        start_time = time.time()

        model.train() # set model to training mode

        train_loss = 0.0
        correct = 0
        total = 0

```

```

# Progress bar for training batches
for images, labels in train_loader:
    images, labels = images.to(device), labels.to(device) # move data to device
    optimizer.zero_grad() # zero the parameter gradients
    outputs = model(images)
    loss = criterion(outputs, labels) # calculate loss

    # Apply regularization if specified
    if reg_type == "L1":
        l1_norm = sum(param.abs().sum() for param in model.parameters())
        loss += reg_lambda * l1_norm
    elif reg_type == "L2":
        l2_norm = sum(param.pow(2).sum() for param in model.parameters())
        loss += reg_lambda * l2_norm

    loss.backward() # backpropagation
    optimizer.step() # update weights
    train_loss += loss.item() # add the loss to the training set loss

    # Calculate training accuracy
    _, predicted = torch.max(outputs, 1) # get predicted class
    total += labels.size(0)
    correct += (predicted == labels).sum().item() # count correct predictions

if step_size is not None and gamma is not None:
    scheduler.step() # update learning rate

# Training set statistics
train_losses.append(round(train_loss / len(train_loader), 4))
train_accuracies.append(round(100 * correct / total, 2) if total > 0 else 0)

# Evaluation on validation set
model.eval() # set model to evaluation mode

val_loss = 0.0
correct = 0
total = 0

with torch.no_grad(): # no need to calculate gradients for validation set
    for images, labels in val_loader:
        images, labels = images.to(device), labels.to(device) # move data to device
        outputs = model(images)
        loss = criterion(outputs, labels) # calculate loss
        val_loss += loss.item() # add the loss to the validation set loss
        _, predicted = torch.max(outputs, 1) # get predicted class
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

# Validation set statistics
val_losses.append(round(val_loss / len(val_loader), 4))
val_accuracies.append(round(100 * correct / total, 2) if total > 0 else 0)

# Print epoch summary
epoch_duration = round(time.time() - start_time)
print(
    f"Epoch {epoch+1}/{n_epochs} | Train Loss: {train_losses[-1]:.4f} (acc. {train_accuracies[-1]:.2f}) | "
    f"Val Loss: {val_losses[-1]:.4f} (acc. {val_accuracies[-1]:.2f}) | Time: {epoch_duration}s"
)

# Calculate and print total training time
total_training_time = round(time.time() - total_start_time)
print(f"Training Time: {total_training_time}s")

# Save model and metrics to file
with open(f"models/{label}.txt", "w") as f:
    model_summary = summary(
        model, input_size=(batch_size, 3, image_size, image_size), verbose=0
    )
    f.write(str(model_summary))
    f.write("\nTraining and Validation Metrics:\n")
    f.write(f"Train Losses: {train_losses}\n")
    f.write(f"Train Accuracies: {train_accuracies}\n")
    f.write(f"Val Losses: {val_losses}\n")
    f.write(f"Val Accuracies: {val_accuracies}\n")

# Save model to file
torch.save(model.state_dict(), f"models/{label}.pth")

```

```

    return {
        "n_epochs": n_epochs,
        "train_losses": train_losses,
        "val_losses": val_losses,
        "train_accuracies": train_accuracies,
        "val_accuracies": val_accuracies,
    }



---


File: result_handler.py

from convolutionalNetwork import ConvolutionalNetwork
from train_model import train_model

def result_handler(configs, device):
    results = {}
    for config in configs:
        model = ConvolutionalNetwork(config["net_config"])
        result = train_model(model, device, config)
        label = config["label"]
        results[label] = result
    return results



---


File: plot_scores.py

import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator

def plot_scores(results):
    for label, data in results.items():
        fig, axes = plt.subplots(1, 2, figsize=(5, 2.5))

        epochs = range(1, data["n_epochs"] + 1)

        # Plot loss
        axes[0].plot(epochs, data["train_losses"], marker="o", markersize=3, label=f"{label} train")
        axes[0].plot(
            epochs,
            data["val_losses"],
            marker="o",
            markersize=3,
            linestyle="--",
            label=f"{label} val",
        )
        axes[0].set_title("loss")
        axes[0].set_xlabel("epoch")
        axes[0].set_ylabel("loss")

        # Plot accuracy
        axes[1].plot(
            epochs, data["train_accuracies"], marker="o", markersize=3, label=f"{label} train"
        )
        axes[1].plot(
            epochs,
            data["val_accuracies"],
            marker="o",
            markersize=3,
            linestyle="--",
            label=f"{label} val",
        )
        axes[1].set_title("accuracy")
        axes[1].set_xlabel("epoch")
        axes[1].set_ylabel("accuracy (%)")

        # Set 4 ticks on the x and y-axis
        axes[0].yaxis.set_major_locator(MaxNLocator(nbins=4))
        axes[1].yaxis.set_major_locator(MaxNLocator(nbins=4))
        axes[0].xaxis.set_major_locator(MaxNLocator(nbins=4))
        axes[1].xaxis.set_major_locator(MaxNLocator(nbins=4))

        # Create a single legend below the plots
        handles, labels = axes[0].get_legend_handles_labels()
        fig.legend(handles, labels, loc="lower center", bbox_to_anchor=(0.5, 0), ncol=2)

```

```

plt.tight_layout(rect=[0, 0.1, 1, 0.9])
plt.show()

File: predict.py

import torch
import matplotlib.pyplot as plt
from collections import defaultdict
from loaders import get_test_loader
from denormalize_image import denormalize_image
from default_config import label_map, test_dir
import csv

def plot_classified_and_misclassified(correctly_classified, misclassified):
    num_images = 3

    # Sort by confidence
    correctly_classified.sort(key=lambda x: x[1], reverse=True)
    misclassified.sort(key=lambda x: x[1], reverse=True)

    # Get top `num_images` for each class
    top_correct = []
    top_incorrect = []
    for label in set(x[2] for x in correctly_classified):
        top_correct.extend([x for x in correctly_classified if x[2] == label][:num_images])
        top_incorrect.extend([x for x in misclassified if x[2] == label][:num_images])

    # Plot correctly and misclassified images
    fig, axs = plt.subplots(2, num_images * 2 + 1, figsize=(25, 10))

    # Add row labels
    axs[0, 0].text(
        0.5, 0.5, "Correctly Classified", fontsize=12, ha="center", va="center", rotation=90
    )
    axs[0, 0].axis("off")
    axs[1, 0].text(0.5, 0.5, "Misclassified", fontsize=12, ha="center", va="center", rotation=90)
    axs[1, 0].axis("off")

    # Show correctly classified images
    for i, (img, prob, true_label, pred_label) in enumerate(top_correct):
        img = denormalize_image(img)
        axs[0, i + 1].imshow(img.permute(1, 2, 0))
        axs[0, i + 1].set_title(f"True: {true_label}, Pred: {pred_label}, Prob: {prob*100:.2f}%")
        axs[0, i + 1].axis("off")

    # Show misclassified images
    for i, (img, prob, true_label, pred_label) in enumerate(top_incorrect):
        img = denormalize_image(img)
        axs[1, i + 1].imshow(img.permute(1, 2, 0))
        axs[1, i + 1].set_title(f"True: {true_label}, Pred: {pred_label}, Prob: {prob*100:.2f}%")
        axs[1, i + 1].axis("off")

    plt.tight_layout()
    plt.show()

def predict(model, device, model_path, results_file="image_probabilities.csv"):
    # Load model
    model.to(device)
    model.load_state_dict(torch.load(model_path, weights_only=True))
    model.eval()

    # Get loader
    test_loader = get_test_loader(test_dir)

    misclassified = []
    correctly_classified = []
    correct_class_counts = defaultdict(int)
    misclass_class_counts = defaultdict(int)

    # Access image paths directly from the dataset within the loader
    image_paths = [sample[0] for sample in test_loader.dataset.samples]

    # Open CSV file for writing

```

```

with open(results_file, mode="w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow([
        "Image Name", "Correct Prediction", "True Label", "Predicted Label", "Probabilities"
    ])

    with torch.no_grad():
        for batch_idx, (images, labels) in enumerate(test_loader):
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)

            probs = torch.softmax(outputs, dim=1)
            predicted_labels = torch.argmax(probs, dim=1)

            for idx in range(images.size(0)):
                image_index = batch_idx * test_loader.batch_size + idx
                img = images[idx].cpu()
                image_name = image_paths[image_index]

                true_label_num = labels[idx].cpu().item()
                pred_label_num = predicted_labels[idx].cpu().item()
                true_label = label_map[true_label_num]
                pred_label = label_map[pred_label_num]

                prob_values = probs[idx].cpu().tolist()
                correct_prediction = true_label == pred_label

                if correct_prediction:
                    correctly_classified.append((img, max(prob_values), true_label, pred_label))
                    correct_class_counts[true_label] += 1
                else:
                    misclassified.append((img, max(prob_values), true_label, pred_label))
                    misclass_class_counts[true_label] += 1

            # Write to CSV
            writer.writerow(
                [
                    image_name,
                    correct_prediction,
                    true_label,
                    pred_label,
                    [f"{p*100:.2f}%" for p in prob_values],
                ]
            )

# Print total counts and accuracy
total_correct = len(correctly_classified)
total_misclassified = len(misclassified)
total = total_correct + total_misclassified
accuracy = round((total_correct / total) * 100, 2)
total_cats = correct_class_counts["cat"] + misclass_class_counts["cat"]
total_dogs = correct_class_counts["dog"] + misclass_class_counts["dog"]
cat_acc = round(correct_class_counts["cat"] / total_cats * 100, 2)
dog_acc = round(correct_class_counts["dog"] / total_dogs * 100, 2)
print(
    f"Total Correctly Classified: {total_correct} | Total Misclassified: {total_misclassified} | Accuracy: {accuracy}%"
)

# Print class-wise statistics
print("Class-wise Correctly Classified Counts:")
for label, count in correct_class_counts.items():
    print(f"- Class {label}: {count}")
print("Class-wise Misclassified Counts:")
for label, count in misclass_class_counts.items():
    print(f"- Class {label}: {count}")
print(f"Cat Accuracy: {cat_acc}% | Dog Accuracy: {dog_acc}%")

plot_classified_and_misclassified(correctly_classified, misclassified)

```

File: plot_individual_feature_maps.py

```

import matplotlib.pyplot as plt
import torch
from PIL import Image
from loaders import get_test_transform

```

```

def plot_individual_feature_maps(
    model, device, img_path, output_file="individual_feature_maps.png"
):
    # Load and preprocess the image
    img = Image.open(img_path).convert("RGB")
    test_transform = get_test_transform()
    img_tensor = (
        test_transform(img).unsqueeze(0).to(device)
    ) # apply transforms and add batch dimension

    # Plot and save the original image
    _, ax = plt.subplots(figsize=(5, 5))
    ax.imshow(img)
    ax.axis("off")
    ax.set_title("Original Image", fontsize=14)
    original_image_file = f"{output_file}_original.png"
    plt.savefig(original_image_file)
    plt.show()

    # Set model to evaluation mode and move to correct device
    model.eval()
    model.to(device)

    # Collect all convolutional layers
    conv_layers = []
    model_children = list(model.children())

    for layer in model_children:
        if isinstance(layer, torch.nn.Conv2d):
            conv_layers.append(layer)
        elif isinstance(layer, torch.nn.Sequential):
            for child in layer.children():
                if isinstance(child, torch.nn.Conv2d):
                    conv_layers.append(child)

    # Forward pass through model and collect feature maps
    outputs = []
    for layer in conv_layers:
        img_tensor = layer(img_tensor)
        outputs.append(img_tensor)

    # Create the figure for feature maps
    for layer_idx, feature_map in enumerate(outputs):
        feature_map = feature_map.squeeze(0).cpu() # remove batch dimension and move to CPU
        num_filters = feature_map.size(0) # number of filters in this layer

        # Determine grid layout for plotting all filters
        col_size = 8 # number of columns
        row_size = (num_filters + col_size - 1) // col_size

        _, axes = plt.subplots(row_size, col_size, figsize=(col_size * 2, row_size * 2))
        axes = axes.flatten()

        # Plot each filter's feature map
        for filter_idx in range(num_filters):
            single_filter_map = feature_map[filter_idx].detach().numpy()
            axes[filter_idx].imshow(single_filter_map, cmap="gray")
            axes[filter_idx].axis("off")
            axes[filter_idx].set_title(f"Filter {filter_idx + 1}", fontsize=8)

        # Turn off any unused subplots
        for idx in range(num_filters, len(axes)):
            axes[idx].axis("off")

        # Save figure for each layer
        plt.tight_layout()
        layer_output_file = f"{output_file}_layer_{layer_idx + 1}.png"
        plt.savefig(layer_output_file)
        plt.show()

```

File: plot_transformations.py

```

import matplotlib.pyplot as plt
import math

```

```
num_images_in_row = 3

def plot_transformations(transformed_images, transforms):
    num_images = len(transformed_images)
    rows = math.ceil(num_images / num_images_in_row)

    _, axes = plt.subplots(
        rows,
        min(num_images_in_row, num_images),
        figsize=(3 * min(num_images_in_row, num_images), 3 * rows),
    )

    if rows == 1:
        axes = [axes]
    elif num_images <= num_images_in_row:
        axes = [axes]

    # Plot each image in the corresponding subplot
    for i, img in enumerate(transformed_images):
        row, col = divmod(i, num_images_in_row)
        axes[row][col].imshow(img)
        axes[row][col].set_title(f"{{transforms[i]}}")

    # Turn off axis for all subplots
    for row in axes:
        for ax in row:
            ax.axis("off")

    plt.tight_layout()
    plt.show()
```

B Notebook

notebook

November 17, 2024

0.1 Libraries

```
[ ]: import os
import random
import matplotlib.pyplot as plt
from PIL import Image

import torch
import torch.nn as nn
from torchvision import datasets, transforms
import torchvision.transforms.functional as TF
from torchvision.models import alexnet, AlexNet_Weights, AlexNet

from default_config import default_config, default_net_config, default_train_config, data_dir, train_dir, u
˓→no_transform_config
from convolutionalNetwork import ConvolutionalNetwork
from predict import predict
from plot_scores import plot_scores
from plot_transformations import plot_transformations
from plot_individual_feature_maps import plot_individual_feature_maps
from denormalize_image import denormalize_image
from loaders import get_train_loader
from result_handler import result_handler
from train_model import train_model
```

0.2 Check device

```
[2]: device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
torch.__version__
```

cuda

[2]: '2.5.1+cu124'

0.3 Seed

```
[3]: seed = 42

random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
```

0.4 Explorative Data Analysis

0.4.1 Dataset walking

```
[4]: for dirname, dirnames, filenames in os.walk(data_dir):
    if len(dirnames) == 0: # only innermost directories
        print(f'{dirname} directory contains {len(filenames)} images')

'data\test\cats' directory contains 200 images
'data\test\dogs' directory contains 200 images
'data\train\cats' directory contains 1000 images
'data\train\dogs' directory contains 1000 images
'data\validation\cats' directory contains 300 images
'data\validation\dogs' directory contains 300 images
```

0.4.2 Labels

```
[5]: print(datasets.ImageFolder(train_dir, transform=transforms.ToTensor()).class_to_idx)
{'cats': 0, 'dogs': 1}
```

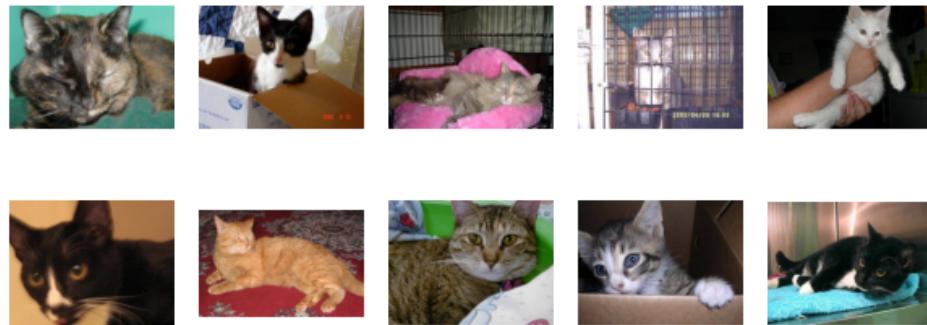
0.4.3 Plot 10 random images of both cats and dogs

```
[6]: train_cats_dir = f"{train_dir}/cats"
train_dogs_dir = f"{train_dir}/dogs"

cats_images = random.sample([f"{train_cats_dir}/{filename}" for filename in os.listdir(train_cats_dir)], 10)
dogs_images = random.sample([f"{train_dogs_dir}/{filename}" for filename in os.listdir(train_dogs_dir)], 10)

# Plot cat images
fig1, axes1 = plt.subplots(2, 5, figsize=(6, 3))
for i, img in enumerate(cats_images):
    axes1[i // 5, i % 5].imshow(plt.imread(img))
    axes1[i // 5, i % 5].axis("off")
plt.tight_layout()
plt.show()

# Plot dog images
fig2, axes2 = plt.subplots(2, 5, figsize=(6, 3))
for i, img in enumerate(dogs_images):
    axes2[i // 5, i % 5].imshow(plt.imread(img))
    axes2[i // 5, i % 5].axis("off")
plt.tight_layout()
plt.show()
```





0.5 Image Size

```
[7]: image = Image.open(cats_images[1])
print(f"Image size: {image.size}")

# Plot image_resized
plt.figure(figsize=(4, 4))
plt.imshow(image)
plt.title("Original Image")
plt.axis("off")
plt.show()
```

Image size: (500, 374)

Original Image



```
[8]: factors = [125, 150, 175, 200, 225, 250]
transformed_images = []
for factor in factors:
```

```

transformed_image = TF.resize(image, factor)
transformed_images.append(transformed_image)

plot_transformations(transformed_images, factors)

```



An image size of 225 seems good, however, for compatibility with pre-trained models we choose 224.

```
[9]: image_resized = TF.resize(image, 224)
```

0.6 Base Model

```

[10]: net_config0 = {**default_net_config,
    "cv_layers": [
        {"out_channels": 32, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": False},
        {"out_channels": 64, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": False},
        {"out_channels": 128, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": False},
        {"out_channels": 256, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": False},
    ],
    "fc_layers": [
        {"out_features": 256, "batch_norm": False, "dropout_rate": 0},
        {"out_features": 128, "batch_norm": False, "dropout_rate": 0}
    ]
}

config0 = [{**default_config, "label": "base_model", "n_epochs": 25, "net_config": net_config0, "transform_config": no_transform_config}]
results = result_handler(config0, device)
plot_scores(results)

```

```

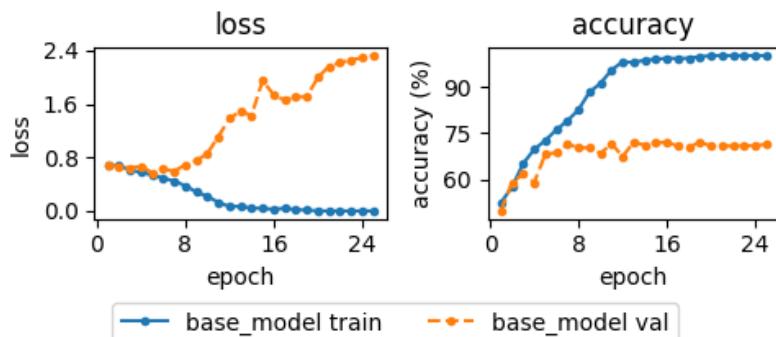
Experiment: base_model
Epoch 1/25 | Train Loss: 0.6952 (acc. 52.65%) | Val Loss: 0.6951 (acc. 50.00%) |
Time: 35s

```

```

Epoch 2/25 | Train Loss: 0.6737 (acc. 57.75%) | Val Loss: 0.6627 (acc. 59.17%) |
Time: 35s
Epoch 3/25 | Train Loss: 0.6153 (acc. 65.20%) | Val Loss: 0.6417 (acc. 61.83%) |
Time: 34s
Epoch 4/25 | Train Loss: 0.5763 (acc. 69.80%) | Val Loss: 0.6698 (acc. 59.17%) |
Time: 34s
Epoch 5/25 | Train Loss: 0.5369 (acc. 72.65%) | Val Loss: 0.5722 (acc. 68.17%) |
Time: 33s
Epoch 6/25 | Train Loss: 0.4914 (acc. 76.10%) | Val Loss: 0.6250 (acc. 68.83%) |
Time: 34s
Epoch 7/25 | Train Loss: 0.4459 (acc. 79.00%) | Val Loss: 0.5959 (acc. 71.50%) |
Time: 34s
Epoch 8/25 | Train Loss: 0.3771 (acc. 82.50%) | Val Loss: 0.6838 (acc. 70.33%) |
Time: 34s
Epoch 9/25 | Train Loss: 0.2845 (acc. 88.25%) | Val Loss: 0.7522 (acc. 70.33%) |
Time: 34s
Epoch 10/25 | Train Loss: 0.2194 (acc. 91.05%) | Val Loss: 0.8570 (acc. 68.50%) |
Time: 33s
Epoch 11/25 | Train Loss: 0.1277 (acc. 95.40%) | Val Loss: 1.1103 (acc. 71.67%) |
Time: 34s
Epoch 12/25 | Train Loss: 0.0657 (acc. 98.00%) | Val Loss: 1.3821 (acc. 67.50%) |
Time: 33s
Epoch 13/25 | Train Loss: 0.0685 (acc. 97.80%) | Val Loss: 1.4903 (acc. 72.17%) |
Time: 33s
Epoch 14/25 | Train Loss: 0.0439 (acc. 98.55%) | Val Loss: 1.4245 (acc. 71.17%) |
Time: 33s
Epoch 15/25 | Train Loss: 0.0373 (acc. 98.80%) | Val Loss: 1.9494 (acc. 71.83%) |
Time: 34s
Epoch 16/25 | Train Loss: 0.0244 (acc. 99.20%) | Val Loss: 1.7371 (acc. 72.17%) |
Time: 34s
Epoch 17/25 | Train Loss: 0.0406 (acc. 99.05%) | Val Loss: 1.6526 (acc. 70.83%) |
Time: 33s
Epoch 18/25 | Train Loss: 0.0201 (acc. 99.20%) | Val Loss: 1.7009 (acc. 70.50%) |
Time: 33s
Epoch 19/25 | Train Loss: 0.0110 (acc. 99.65%) | Val Loss: 1.7072 (acc. 72.17%) |
Time: 33s
Epoch 20/25 | Train Loss: 0.0020 (acc. 100.00%) | Val Loss: 1.9952 (acc. 70.83%) |
Time: 34s
Epoch 21/25 | Train Loss: 0.0004 (acc. 100.00%) | Val Loss: 2.1588 (acc. 71.00%) |
Time: 33s
Epoch 22/25 | Train Loss: 0.0002 (acc. 100.00%) | Val Loss: 2.2134 (acc. 70.83%) |
Time: 33s
Epoch 23/25 | Train Loss: 0.0001 (acc. 100.00%) | Val Loss: 2.2571 (acc. 71.00%) |
Time: 34s
Epoch 24/25 | Train Loss: 0.0001 (acc. 100.00%) | Val Loss: 2.2897 (acc. 71.00%) |
Time: 33s
Epoch 25/25 | Train Loss: 0.0001 (acc. 100.00%) | Val Loss: 2.3216 (acc. 71.50%) |
Time: 33s
Training Time: 840s

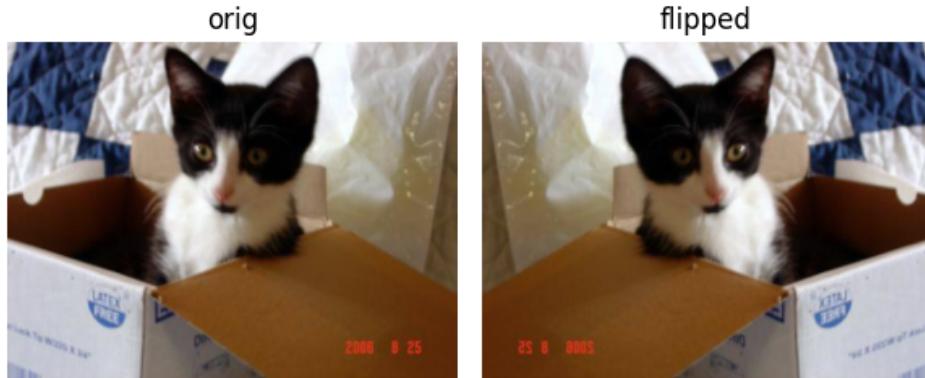
```



0.7 Data Augmentation

0.7.1 Flip

```
[11]: transformed_images = []
transformed_images.append(image_resized)
transformed_images.append(tf.hflip(image_resized))
plot_transformations(transformed_images, ["orig", "flipped"])
```



```
[12]: transformed_images = []
transformed_images.append(image_resized)
transformed_images.append(tf.vflip(image_resized))
plot_transformations(transformed_images, ["orig", "flipped"])
```



Choose to only horizontal flip.

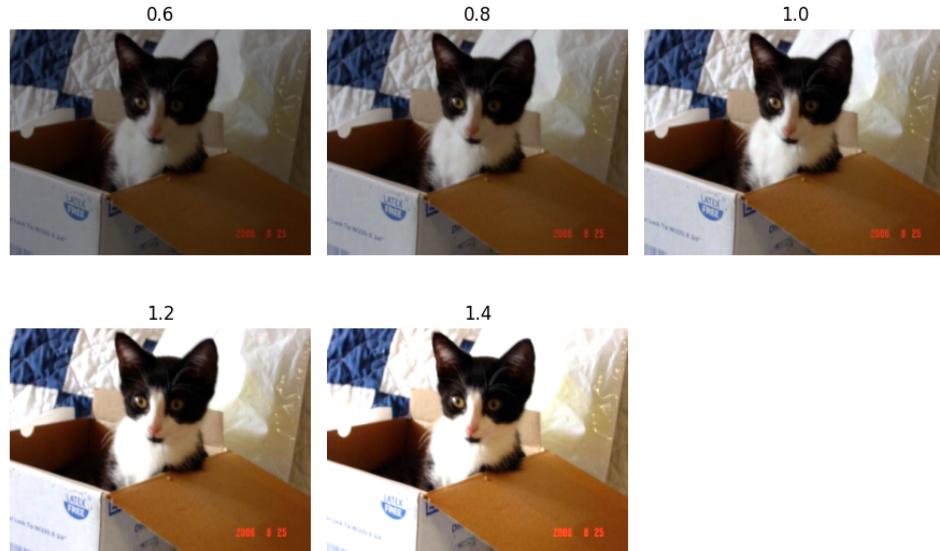
0.7.2 Image color

Brightness

```
[13]: factors = [0.6, 0.8, 1.0, 1.2, 1.4]
```

```
transformed_images = []
for factor in factors:
    transformed_image = TF.adjust_brightness(image_resized, factor)
    transformed_images.append(transformed_image)

plot_transformations(transformed_images, factors)
```

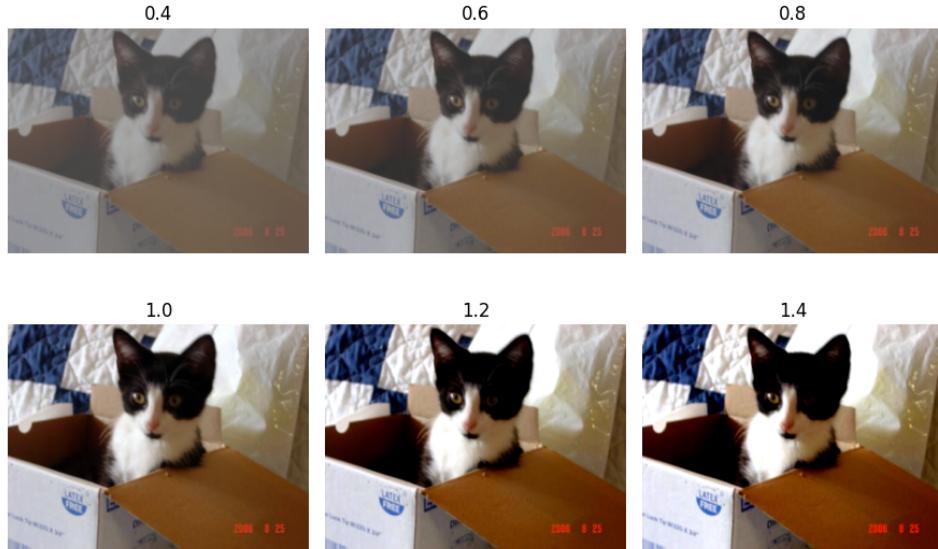


Contrast

```
[14]: factors = [0.4, 0.6, 0.8, 1.0, 1.2, 1.4]

transformed_images = []
for factor in factors:
    transformed_image = TF.adjust_contrast(image_resized, factor)
    transformed_images.append(transformed_image)

plot_transformations(transformed_images, factors)
```



```
Saturation
[15]: factors = [0.2, 0.8, 1.0, 1.2, 1.6, 2.0]

transformed_images = []
for factor in factors:
    transformed_image = TF.adjust_saturation(image_resized, factor)
    transformed_images.append(transformed_image)

plot_transformations(transformed_images, factors)
```



Hue

```
[16]: factors = [0.2, 0.1, 0, -0.1, -0.2]

transformed_images = []
for factor in factors:
    transformed_image = TF.adjust_hue(image_resized, factor)
    transformed_images.append(transformed_image)

plot_transformations(transformed_images, factors)
```

**0.7.3 Affine (rotation, scaling, translation, shearing)**

```
[17]: def apply_affine(image, angle, translate=(0, 0), scale=1.0, shear=0):
    return TF.affine(image, angle=angle, translate=translate, scale=scale, shear=shear)

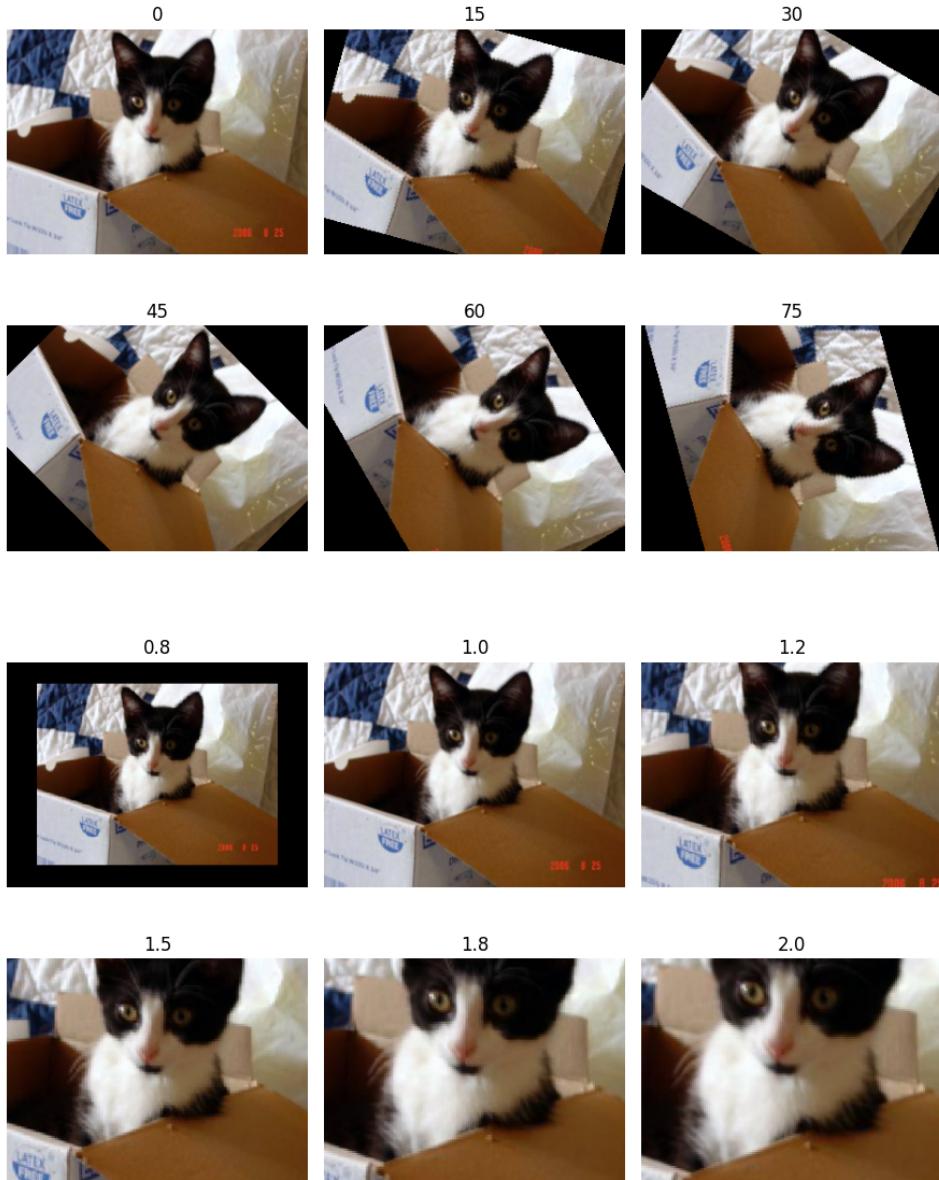
# Rotation
angles = [0, 15, 30, 45, 60, 75]
transformed_images = [apply_affine(image_resized, angle=angle) for angle in angles]
plot_transformations(transformed_images, angles)

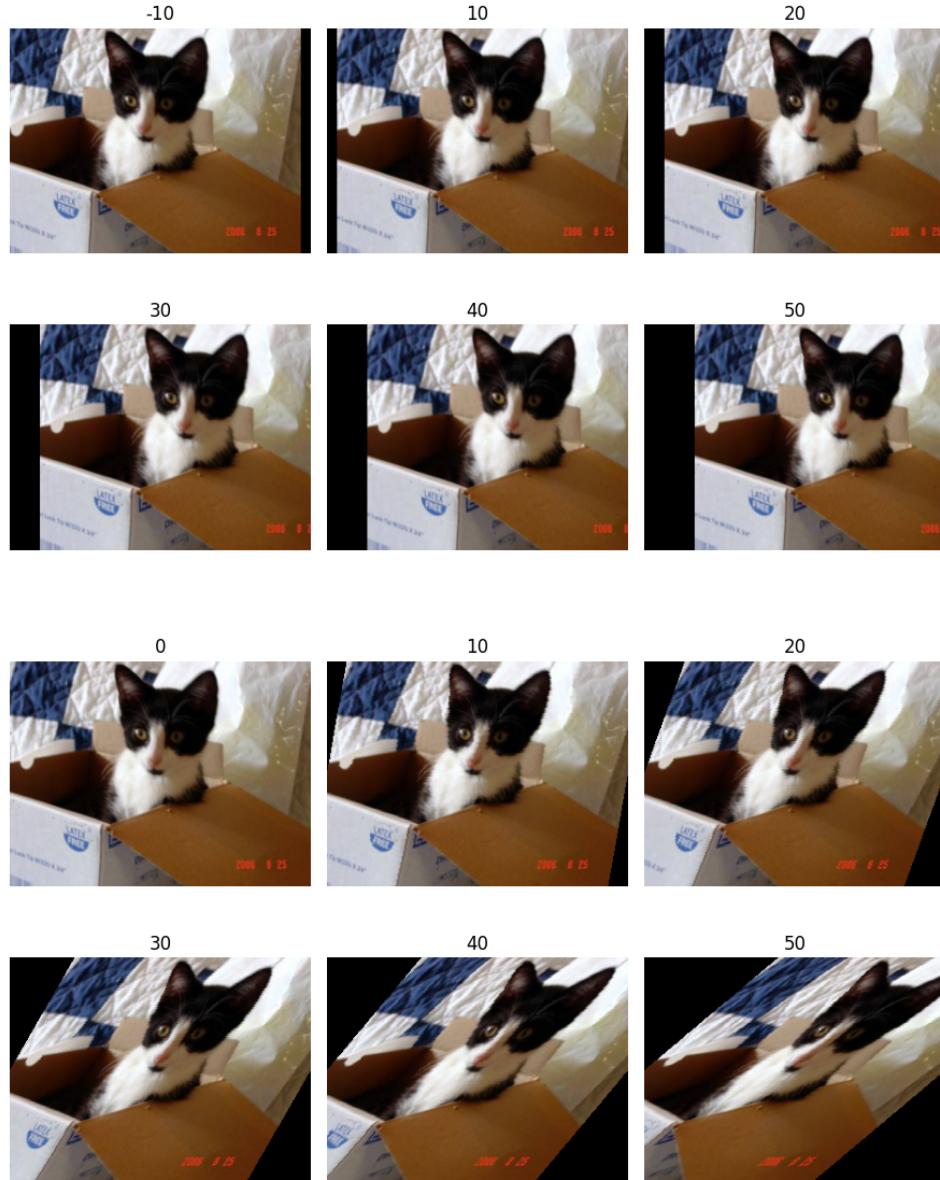
# Scales
scales = [0.8, 1.0, 1.2, 1.5, 1.8, 2.0]
transformed_images = [apply_affine(image_resized, angle=0, scale=scale) for scale in scales]
plot_transformations(transformed_images, scales)

# Translations
translations = [(-10, 0), (10, 0), (20, 0), (30, 0), (40, 0), (50, 0)]
transformed_images = [apply_affine(image_resized, angle=0, translate=translate) for translate in translations]
plot_transformations(transformed_images, [t[0] for t in translations])

# Shear
shears = [0, 10, 20, 30, 40, 50]
```

```
transformed_images = [apply_affine(image_resized, angle=0, shear=shear) for shear in shears]
plot_transformations(transformed_images, shears)
```





0.7.4 Final loader

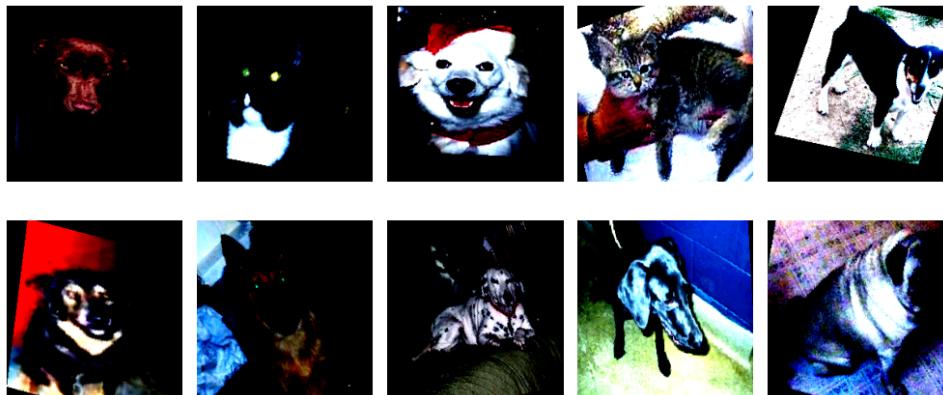
With normalization

```
[18]: train_loader = get_train_loader()
images, _ = next(iter(train_loader))
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for i in range(10):
    img = images[i].permute(1, 2, 0).cpu().numpy() # convert to (H, W, C)
```

```

    img = img.clip(0, 1) # clip values to [0, 1]
    axes[i // 5, i % 5].imshow(img)
    axes[i // 5, i % 5].axis("off")
plt.tight_layout()
plt.show()

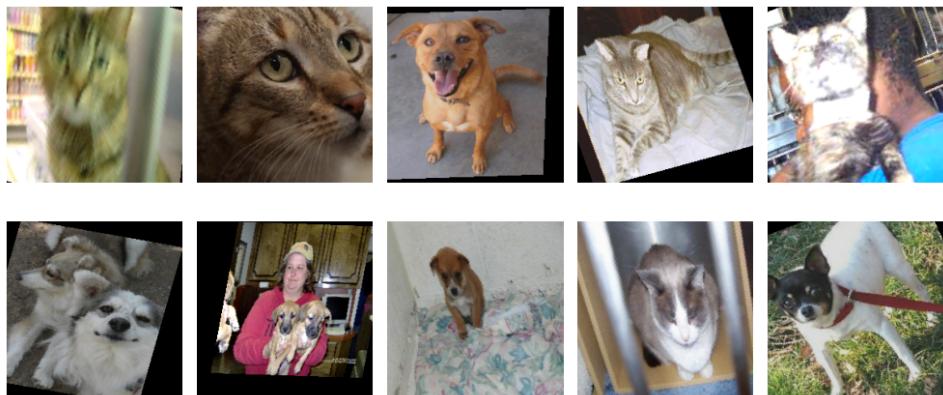
```

**Without normalization**

```

[19]: train_loader = get_train_loader()
images, _ = next(iter(train_loader))
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for i in range(10):
    axes[i // 5, i % 5].imshow(denormalize_image(images[i]).permute(1, 2, 0)) # denormalize and convert to
    ↪(H, W, C)
    axes[i // 5, i % 5].axis("off")
plt.tight_layout()
plt.show()

```



0.8 With Data Augmentation

```
[20]: config0 = [{**default_config, "label": "w_augmentation", "net_config": net_config0}]
results = result_handler(config0, device)
plot_scores(results)
```

```
Experiment: w_augmentation
Epoch 1/120 | Train Loss: 0.6994 (acc. 51.70%) | Val Loss: 0.6878 (acc. 60.17%)
| Time: 34s
Epoch 2/120 | Train Loss: 0.6895 (acc. 53.75%) | Val Loss: 0.6895 (acc. 50.33%)
| Time: 34s
Epoch 3/120 | Train Loss: 0.6801 (acc. 54.30%) | Val Loss: 0.6847 (acc. 55.33%)
| Time: 34s
Epoch 4/120 | Train Loss: 0.6598 (acc. 59.65%) | Val Loss: 0.6549 (acc. 60.83%)
| Time: 33s
Epoch 5/120 | Train Loss: 0.6371 (acc. 64.40%) | Val Loss: 0.6706 (acc. 61.67%)
| Time: 34s
Epoch 6/120 | Train Loss: 0.6208 (acc. 65.10%) | Val Loss: 0.6582 (acc. 62.33%)
| Time: 34s
Epoch 7/120 | Train Loss: 0.6096 (acc. 66.50%) | Val Loss: 0.6502 (acc. 65.00%)
| Time: 34s
Epoch 8/120 | Train Loss: 0.5936 (acc. 67.80%) | Val Loss: 0.6547 (acc. 66.00%)
| Time: 34s
Epoch 9/120 | Train Loss: 0.5776 (acc. 69.35%) | Val Loss: 0.5995 (acc. 68.83%)
| Time: 33s
Epoch 10/120 | Train Loss: 0.5514 (acc. 70.85%) | Val Loss: 0.6134 (acc. 68.83%)
| Time: 34s
Epoch 11/120 | Train Loss: 0.5418 (acc. 71.05%) | Val Loss: 0.5918 (acc. 71.00%)
| Time: 34s
Epoch 12/120 | Train Loss: 0.5302 (acc. 72.05%) | Val Loss: 0.5703 (acc. 71.17%)
| Time: 34s
Epoch 13/120 | Train Loss: 0.5586 (acc. 70.55%) | Val Loss: 0.5582 (acc. 71.33%)
| Time: 34s
Epoch 14/120 | Train Loss: 0.5514 (acc. 71.90%) | Val Loss: 0.5450 (acc. 73.17%)
| Time: 34s
Epoch 15/120 | Train Loss: 0.5221 (acc. 73.55%) | Val Loss: 0.5334 (acc. 72.33%)
| Time: 34s
Epoch 16/120 | Train Loss: 0.4971 (acc. 76.05%) | Val Loss: 0.5384 (acc. 75.00%)
| Time: 34s
Epoch 17/120 | Train Loss: 0.5000 (acc. 76.45%) | Val Loss: 0.5475 (acc. 72.83%)
| Time: 34s
Epoch 18/120 | Train Loss: 0.4822 (acc. 77.20%) | Val Loss: 0.5633 (acc. 72.33%)
| Time: 34s
Epoch 19/120 | Train Loss: 0.4796 (acc. 77.05%) | Val Loss: 0.5226 (acc. 77.83%)
| Time: 34s
Epoch 20/120 | Train Loss: 0.4964 (acc. 76.40%) | Val Loss: 0.5302 (acc. 74.00%)
| Time: 34s
Epoch 21/120 | Train Loss: 0.4484 (acc. 78.90%) | Val Loss: 0.6042 (acc. 74.00%)
| Time: 34s
Epoch 22/120 | Train Loss: 0.4650 (acc. 79.00%) | Val Loss: 0.5727 (acc. 70.17%)
| Time: 34s
Epoch 23/120 | Train Loss: 0.4698 (acc. 77.70%) | Val Loss: 0.5292 (acc. 75.83%)
| Time: 33s
Epoch 24/120 | Train Loss: 0.4604 (acc. 78.50%) | Val Loss: 0.4854 (acc. 77.67%)
| Time: 34s
Epoch 25/120 | Train Loss: 0.4516 (acc. 78.15%) | Val Loss: 0.5270 (acc. 76.00%)
| Time: 34s
Epoch 26/120 | Train Loss: 0.4324 (acc. 79.65%) | Val Loss: 0.4821 (acc. 77.17%)
| Time: 34s
Epoch 27/120 | Train Loss: 0.4048 (acc. 80.90%) | Val Loss: 0.4799 (acc. 79.83%)
| Time: 34s
Epoch 28/120 | Train Loss: 0.4169 (acc. 80.40%) | Val Loss: 0.5001 (acc. 80.50%)
| Time: 34s
Epoch 29/120 | Train Loss: 0.4174 (acc. 79.10%) | Val Loss: 0.4819 (acc. 80.00%)
| Time: 34s
Epoch 30/120 | Train Loss: 0.3744 (acc. 83.40%) | Val Loss: 0.5007 (acc. 80.67%)
| Time: 34s
```

```

Epoch 31/120 | Train Loss: 0.3946 (acc. 82.45%) | Val Loss: 0.5338 (acc. 75.83%)
| Time: 34s
Epoch 32/120 | Train Loss: 0.3896 (acc. 82.05%) | Val Loss: 0.4811 (acc. 79.00%)
| Time: 34s
Epoch 33/120 | Train Loss: 0.3570 (acc. 83.80%) | Val Loss: 0.4761 (acc. 80.67%)
| Time: 34s
Epoch 34/120 | Train Loss: 0.3720 (acc. 83.80%) | Val Loss: 0.5233 (acc. 81.00%)
| Time: 34s
Epoch 35/120 | Train Loss: 0.3777 (acc. 83.05%) | Val Loss: 0.4648 (acc. 78.00%)
| Time: 34s
Epoch 36/120 | Train Loss: 0.3590 (acc. 84.55%) | Val Loss: 0.4369 (acc. 82.83%)
| Time: 34s
Epoch 37/120 | Train Loss: 0.3132 (acc. 85.80%) | Val Loss: 0.5858 (acc. 78.67%)
| Time: 34s
Epoch 38/120 | Train Loss: 0.3517 (acc. 83.70%) | Val Loss: 0.4601 (acc. 81.67%)
| Time: 34s
Epoch 39/120 | Train Loss: 0.3332 (acc. 84.80%) | Val Loss: 0.4874 (acc. 82.33%)
| Time: 34s
Epoch 40/120 | Train Loss: 0.3167 (acc. 85.80%) | Val Loss: 0.4798 (acc. 80.83%)
| Time: 33s
Epoch 41/120 | Train Loss: 0.3366 (acc. 84.85%) | Val Loss: 0.4495 (acc. 80.50%)
| Time: 34s
Epoch 42/120 | Train Loss: 0.3023 (acc. 86.95%) | Val Loss: 0.4302 (acc. 83.50%)
| Time: 34s
Epoch 43/120 | Train Loss: 0.3226 (acc. 86.00%) | Val Loss: 0.5216 (acc. 77.67%)
| Time: 34s
Epoch 44/120 | Train Loss: 0.3410 (acc. 85.15%) | Val Loss: 0.4522 (acc. 81.00%)
| Time: 34s
Epoch 45/120 | Train Loss: 0.2803 (acc. 87.95%) | Val Loss: 0.5553 (acc. 80.50%)
| Time: 34s
Epoch 46/120 | Train Loss: 0.2632 (acc. 88.60%) | Val Loss: 0.4526 (acc. 83.83%)
| Time: 34s
Epoch 47/120 | Train Loss: 0.2690 (acc. 88.90%) | Val Loss: 0.4936 (acc. 83.00%)
| Time: 34s
Epoch 48/120 | Train Loss: 0.2800 (acc. 87.25%) | Val Loss: 0.7044 (acc. 80.50%)
| Time: 34s
Epoch 49/120 | Train Loss: 0.3054 (acc. 85.70%) | Val Loss: 0.4693 (acc. 80.83%)
| Time: 34s
Epoch 50/120 | Train Loss: 0.2540 (acc. 88.55%) | Val Loss: 0.4721 (acc. 83.67%)
| Time: 34s
Epoch 51/120 | Train Loss: 0.2448 (acc. 89.65%) | Val Loss: 0.5040 (acc. 83.17%)
| Time: 34s
Epoch 52/120 | Train Loss: 0.2511 (acc. 89.60%) | Val Loss: 0.4375 (acc. 84.83%)
| Time: 34s
Epoch 53/120 | Train Loss: 0.2566 (acc. 89.35%) | Val Loss: 0.5014 (acc. 82.17%)
| Time: 33s
Epoch 54/120 | Train Loss: 0.2481 (acc. 89.65%) | Val Loss: 0.4713 (acc. 85.67%)
| Time: 34s
Epoch 55/120 | Train Loss: 0.2428 (acc. 90.40%) | Val Loss: 0.5083 (acc. 84.83%)
| Time: 34s
Epoch 56/120 | Train Loss: 0.2408 (acc. 89.80%) | Val Loss: 0.4770 (acc. 84.83%)
| Time: 34s
Epoch 57/120 | Train Loss: 0.2381 (acc. 89.55%) | Val Loss: 0.4902 (acc. 84.33%)
| Time: 34s
Epoch 58/120 | Train Loss: 0.2443 (acc. 89.10%) | Val Loss: 0.4495 (acc. 85.17%)
| Time: 34s
Epoch 59/120 | Train Loss: 0.2241 (acc. 90.75%) | Val Loss: 0.5438 (acc. 83.83%)
| Time: 34s
Epoch 60/120 | Train Loss: 0.2040 (acc. 91.30%) | Val Loss: 0.4785 (acc. 83.17%)
| Time: 34s
Epoch 61/120 | Train Loss: 0.1963 (acc. 91.95%) | Val Loss: 0.4777 (acc. 84.67%)
| Time: 34s
Epoch 62/120 | Train Loss: 0.1884 (acc. 91.75%) | Val Loss: 0.4696 (acc. 85.67%)
| Time: 33s
Epoch 63/120 | Train Loss: 0.1929 (acc. 92.90%) | Val Loss: 0.4613 (acc. 82.33%)
| Time: 34s
Epoch 64/120 | Train Loss: 0.2146 (acc. 91.05%) | Val Loss: 0.4857 (acc. 83.50%)
| Time: 33s

```

```

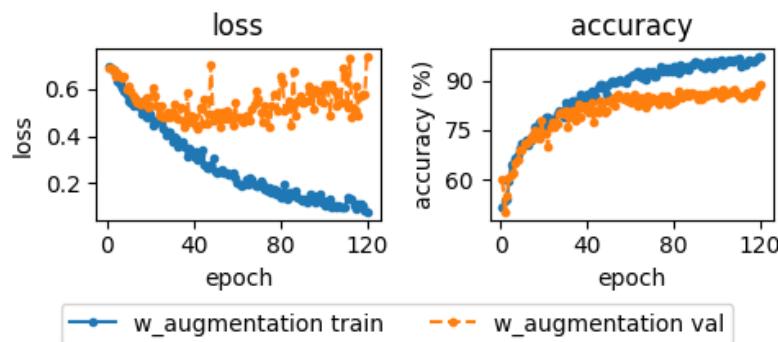
Epoch 65/120 | Train Loss: 0.1977 (acc. 92.10%) | Val Loss: 0.4766 (acc. 84.67%)
| Time: 34s
Epoch 66/120 | Train Loss: 0.2258 (acc. 90.85%) | Val Loss: 0.4648 (acc. 82.67%)
| Time: 34s
Epoch 67/120 | Train Loss: 0.2171 (acc. 90.50%) | Val Loss: 0.5159 (acc. 82.00%)
| Time: 33s
Epoch 68/120 | Train Loss: 0.1954 (acc. 92.65%) | Val Loss: 0.5347 (acc. 85.33%)
| Time: 34s
Epoch 69/120 | Train Loss: 0.1857 (acc. 92.35%) | Val Loss: 0.5925 (acc. 83.33%)
| Time: 34s
Epoch 70/120 | Train Loss: 0.1920 (acc. 92.65%) | Val Loss: 0.5608 (acc. 85.00%)
| Time: 34s
Epoch 71/120 | Train Loss: 0.2130 (acc. 91.45%) | Val Loss: 0.5002 (acc. 84.33%)
| Time: 34s
Epoch 72/120 | Train Loss: 0.1979 (acc. 91.45%) | Val Loss: 0.4596 (acc. 84.00%)
| Time: 34s
Epoch 73/120 | Train Loss: 0.1767 (acc. 92.95%) | Val Loss: 0.5268 (acc. 83.00%)
| Time: 34s
Epoch 74/120 | Train Loss: 0.1656 (acc. 94.20%) | Val Loss: 0.4926 (acc. 84.83%)
| Time: 34s
Epoch 75/120 | Train Loss: 0.1819 (acc. 92.65%) | Val Loss: 0.4421 (acc. 85.17%)
| Time: 34s
Epoch 76/120 | Train Loss: 0.1640 (acc. 93.85%) | Val Loss: 0.5294 (acc. 83.67%)
| Time: 34s
Epoch 77/120 | Train Loss: 0.1723 (acc. 92.85%) | Val Loss: 0.5780 (acc. 85.17%)
| Time: 34s
Epoch 78/120 | Train Loss: 0.1559 (acc. 93.60%) | Val Loss: 0.5791 (acc. 82.83%)
| Time: 34s
Epoch 79/120 | Train Loss: 0.1657 (acc. 92.95%) | Val Loss: 0.5074 (acc. 83.50%)
| Time: 34s
Epoch 80/120 | Train Loss: 0.1373 (acc. 94.05%) | Val Loss: 0.6580 (acc. 84.17%)
| Time: 34s
Epoch 81/120 | Train Loss: 0.1428 (acc. 94.15%) | Val Loss: 0.6294 (acc. 83.67%)
| Time: 34s
Epoch 82/120 | Train Loss: 0.1931 (acc. 92.45%) | Val Loss: 0.5093 (acc. 86.67%)
| Time: 34s
Epoch 83/120 | Train Loss: 0.1341 (acc. 94.80%) | Val Loss: 0.5239 (acc. 85.33%)
| Time: 33s
Epoch 84/120 | Train Loss: 0.1632 (acc. 93.45%) | Val Loss: 0.5531 (acc. 85.00%)
| Time: 34s
Epoch 85/120 | Train Loss: 0.1598 (acc. 93.85%) | Val Loss: 0.4482 (acc. 85.67%)
| Time: 34s
Epoch 86/120 | Train Loss: 0.1448 (acc. 94.25%) | Val Loss: 0.6787 (acc. 86.33%)
| Time: 34s
Epoch 87/120 | Train Loss: 0.1676 (acc. 93.40%) | Val Loss: 0.4874 (acc. 85.17%)
| Time: 34s
Epoch 88/120 | Train Loss: 0.1412 (acc. 94.55%) | Val Loss: 0.5527 (acc. 86.17%)
| Time: 34s
Epoch 89/120 | Train Loss: 0.1353 (acc. 94.55%) | Val Loss: 0.5746 (acc. 85.67%)
| Time: 34s
Epoch 90/120 | Train Loss: 0.1245 (acc. 95.55%) | Val Loss: 0.5443 (acc. 85.67%)
| Time: 34s
Epoch 91/120 | Train Loss: 0.1351 (acc. 95.00%) | Val Loss: 0.5426 (acc. 84.17%)
| Time: 35s
Epoch 92/120 | Train Loss: 0.1204 (acc. 95.25%) | Val Loss: 0.5516 (acc. 86.00%)
| Time: 34s
Epoch 93/120 | Train Loss: 0.1281 (acc. 94.30%) | Val Loss: 0.6152 (acc. 84.17%)
| Time: 34s
Epoch 94/120 | Train Loss: 0.1511 (acc. 93.95%) | Val Loss: 0.5863 (acc. 86.00%)
| Time: 34s
Epoch 95/120 | Train Loss: 0.1466 (acc. 94.25%) | Val Loss: 0.5165 (acc. 85.17%)
| Time: 34s
Epoch 96/120 | Train Loss: 0.1164 (acc. 95.45%) | Val Loss: 0.5992 (acc. 86.00%)
| Time: 33s
Epoch 97/120 | Train Loss: 0.1358 (acc. 94.60%) | Val Loss: 0.5839 (acc. 85.67%)
| Time: 34s
Epoch 98/120 | Train Loss: 0.1526 (acc. 93.85%) | Val Loss: 0.5531 (acc. 86.67%)
| Time: 34s

```

```

Epoch 99/120 | Train Loss: 0.1201 (acc. 95.60%) | Val Loss: 0.5915 (acc. 86.67%)
| Time: 34s
Epoch 100/120 | Train Loss: 0.1278 (acc. 94.75%) | Val Loss: 0.5410 (acc.
86.50%) | Time: 34s
Epoch 101/120 | Train Loss: 0.1037 (acc. 95.50%) | Val Loss: 0.6229 (acc.
85.83%) | Time: 34s
Epoch 102/120 | Train Loss: 0.1052 (acc. 96.05%) | Val Loss: 0.6190 (acc.
84.67%) | Time: 34s
Epoch 103/120 | Train Loss: 0.1343 (acc. 94.55%) | Val Loss: 0.4870 (acc.
87.17%) | Time: 34s
Epoch 104/120 | Train Loss: 0.1016 (acc. 95.80%) | Val Loss: 0.6222 (acc.
85.67%) | Time: 34s
Epoch 105/120 | Train Loss: 0.1025 (acc. 95.90%) | Val Loss: 0.5933 (acc.
86.33%) | Time: 34s
Epoch 106/120 | Train Loss: 0.1005 (acc. 96.00%) | Val Loss: 0.5320 (acc.
86.50%) | Time: 34s
Epoch 107/120 | Train Loss: 0.1053 (acc. 95.80%) | Val Loss: 0.5832 (acc.
86.67%) | Time: 34s
Epoch 108/120 | Train Loss: 0.0956 (acc. 96.05%) | Val Loss: 0.5449 (acc.
87.33%) | Time: 34s
Epoch 109/120 | Train Loss: 0.0990 (acc. 96.30%) | Val Loss: 0.6869 (acc.
86.33%) | Time: 34s
Epoch 110/120 | Train Loss: 0.1009 (acc. 96.60%) | Val Loss: 0.6611 (acc.
85.50%) | Time: 34s
Epoch 111/120 | Train Loss: 0.1411 (acc. 94.25%) | Val Loss: 0.5240 (acc.
85.00%) | Time: 34s
Epoch 112/120 | Train Loss: 0.1319 (acc. 94.55%) | Val Loss: 0.7344 (acc.
84.67%) | Time: 34s
Epoch 113/120 | Train Loss: 0.1309 (acc. 94.85%) | Val Loss: 0.4805 (acc.
86.00%) | Time: 34s
Epoch 114/120 | Train Loss: 0.1069 (acc. 95.80%) | Val Loss: 0.5167 (acc.
86.50%) | Time: 34s
Epoch 115/120 | Train Loss: 0.0910 (acc. 96.20%) | Val Loss: 0.6112 (acc.
86.17%) | Time: 34s
Epoch 116/120 | Train Loss: 0.1115 (acc. 95.60%) | Val Loss: 0.4870 (acc.
87.00%) | Time: 34s
Epoch 117/120 | Train Loss: 0.1119 (acc. 95.45%) | Val Loss: 0.5688 (acc.
85.17%) | Time: 34s
Epoch 118/120 | Train Loss: 0.1028 (acc. 96.25%) | Val Loss: 0.5792 (acc.
86.50%) | Time: 34s
Epoch 119/120 | Train Loss: 0.0835 (acc. 96.95%) | Val Loss: 0.5764 (acc.
88.50%) | Time: 34s
Epoch 120/120 | Train Loss: 0.0785 (acc. 97.15%) | Val Loss: 0.7381 (acc.
88.83%) | Time: 34s
Training Time: 4035s

```



0.9 Regularization

```
[21]: net_config1 = {**default_net_config,
    "cv_layers": [
        {"out_channels": 32, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
        {"out_channels": 64, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
        {"out_channels": 128, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
        {"out_channels": 256, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
    ],
    "fc_layers": [
        {"out_features": 256, "batch_norm": True, "dropout_rate": 0.4},
        {"out_features": 128, "batch_norm": True, "dropout_rate": 0.2},
    ]
}

train_config1 = {**default_train_config, "step_size": 25, "gamma": 0.5, "weight_decay": 1e-4}

configs1 = [
    {**default_config, "label": "reg_1", "net_config": net_config1},
    {**default_config, "label": "reg_2", "net_config": net_config1, "train_config": train_config1},
]

results = result_handler(configs1, device)
plot_scores(results)
```

Experiment: reg_1
Epoch 1/120 | Train Loss: 0.6962 (acc. 56.95%) | Val Loss: 0.8926 (acc. 54.83%)
| Time: 34s
Epoch 2/120 | Train Loss: 0.6597 (acc. 58.95%) | Val Loss: 0.6464 (acc. 63.17%)
| Time: 34s
Epoch 3/120 | Train Loss: 0.6288 (acc. 64.65%) | Val Loss: 0.6925 (acc. 56.33%)
| Time: 34s
Epoch 4/120 | Train Loss: 0.5979 (acc. 67.80%) | Val Loss: 0.6143 (acc. 61.50%)
| Time: 34s
Epoch 5/120 | Train Loss: 0.5828 (acc. 68.55%) | Val Loss: 0.6010 (acc. 70.17%)
| Time: 34s
Epoch 6/120 | Train Loss: 0.5800 (acc. 68.15%) | Val Loss: 0.6204 (acc. 69.33%)
| Time: 34s
Epoch 7/120 | Train Loss: 0.5622 (acc. 70.75%) | Val Loss: 0.6065 (acc. 68.17%)
| Time: 34s
Epoch 8/120 | Train Loss: 0.5561 (acc. 72.35%) | Val Loss: 0.5784 (acc. 71.33%)
| Time: 34s
Epoch 9/120 | Train Loss: 0.5447 (acc. 72.15%) | Val Loss: 0.5474 (acc. 71.67%)
| Time: 34s
Epoch 10/120 | Train Loss: 0.5238 (acc. 73.80%) | Val Loss: 0.6497 (acc. 66.00%)
| Time: 34s
Epoch 11/120 | Train Loss: 0.5209 (acc. 74.15%) | Val Loss: 0.5724 (acc. 71.17%)
| Time: 34s
Epoch 12/120 | Train Loss: 0.5160 (acc. 73.65%) | Val Loss: 0.6286 (acc. 67.50%)
| Time: 34s
Epoch 13/120 | Train Loss: 0.5104 (acc. 74.75%) | Val Loss: 0.6504 (acc. 68.50%)
| Time: 34s
Epoch 14/120 | Train Loss: 0.4879 (acc. 75.85%) | Val Loss: 0.5559 (acc. 73.50%)
| Time: 34s
Epoch 15/120 | Train Loss: 0.4844 (acc. 76.60%) | Val Loss: 0.5910 (acc. 72.17%)
| Time: 34s
Epoch 16/120 | Train Loss: 0.4893 (acc. 76.20%) | Val Loss: 0.6220 (acc. 70.50%)
| Time: 34s
Epoch 17/120 | Train Loss: 0.4809 (acc. 77.30%) | Val Loss: 0.5460 (acc. 71.00%)
| Time: 34s
Epoch 18/120 | Train Loss: 0.4896 (acc. 76.25%) | Val Loss: 0.5584 (acc. 71.00%)
| Time: 34s
Epoch 19/120 | Train Loss: 0.4742 (acc. 77.50%) | Val Loss: 0.5032 (acc. 75.67%)

```

| Time: 34s
Epoch 20/120 | Train Loss: 0.4576 (acc. 78.55%) | Val Loss: 0.4965 (acc. 75.83%)
| Time: 34s
Epoch 21/120 | Train Loss: 0.4478 (acc. 78.45%) | Val Loss: 0.5090 (acc. 75.33%)
| Time: 34s
Epoch 22/120 | Train Loss: 0.4220 (acc. 81.10%) | Val Loss: 0.5131 (acc. 77.83%)
| Time: 34s
Epoch 23/120 | Train Loss: 0.4385 (acc. 79.40%) | Val Loss: 0.5742 (acc. 71.17%)
| Time: 34s
Epoch 24/120 | Train Loss: 0.4452 (acc. 78.00%) | Val Loss: 0.5460 (acc. 72.67%)
| Time: 34s
Epoch 25/120 | Train Loss: 0.4184 (acc. 81.15%) | Val Loss: 0.4871 (acc. 77.00%)
| Time: 34s
Epoch 26/120 | Train Loss: 0.4355 (acc. 80.25%) | Val Loss: 0.4897 (acc. 77.67%)
| Time: 34s
Epoch 27/120 | Train Loss: 0.4517 (acc. 79.40%) | Val Loss: 0.4839 (acc. 75.33%)
| Time: 34s
Epoch 28/120 | Train Loss: 0.4137 (acc. 82.15%) | Val Loss: 0.4873 (acc. 76.67%)
| Time: 34s
Epoch 29/120 | Train Loss: 0.4234 (acc. 80.20%) | Val Loss: 0.6243 (acc. 69.50%)
| Time: 34s
Epoch 30/120 | Train Loss: 0.4175 (acc. 81.00%) | Val Loss: 0.4784 (acc. 78.00%)
| Time: 34s
Epoch 31/120 | Train Loss: 0.3980 (acc. 81.00%) | Val Loss: 0.4372 (acc. 80.67%)
| Time: 34s
Epoch 32/120 | Train Loss: 0.3954 (acc. 81.20%) | Val Loss: 0.5249 (acc. 76.83%)
| Time: 34s
Epoch 33/120 | Train Loss: 0.3764 (acc. 82.85%) | Val Loss: 0.4968 (acc. 76.67%)
| Time: 34s
Epoch 34/120 | Train Loss: 0.3727 (acc. 84.40%) | Val Loss: 0.4571 (acc. 80.17%)
| Time: 34s
Epoch 35/120 | Train Loss: 0.3894 (acc. 82.70%) | Val Loss: 0.4526 (acc. 79.33%)
| Time: 34s
Epoch 36/120 | Train Loss: 0.3736 (acc. 82.20%) | Val Loss: 0.4761 (acc. 78.33%)
| Time: 34s
Epoch 37/120 | Train Loss: 0.3521 (acc. 84.20%) | Val Loss: 0.4387 (acc. 81.00%)
| Time: 34s
Epoch 38/120 | Train Loss: 0.3841 (acc. 82.60%) | Val Loss: 0.4475 (acc. 80.83%)
| Time: 34s
Epoch 39/120 | Train Loss: 0.3636 (acc. 84.70%) | Val Loss: 0.4355 (acc. 80.50%)
| Time: 34s
Epoch 40/120 | Train Loss: 0.3646 (acc. 84.25%) | Val Loss: 0.4286 (acc. 80.67%)
| Time: 34s
Epoch 41/120 | Train Loss: 0.3341 (acc. 85.10%) | Val Loss: 0.4978 (acc. 79.33%)
| Time: 34s
Epoch 42/120 | Train Loss: 0.3456 (acc. 85.15%) | Val Loss: 0.4592 (acc. 81.50%)
| Time: 34s
Epoch 43/120 | Train Loss: 0.3361 (acc. 86.00%) | Val Loss: 0.6471 (acc. 70.50%)
| Time: 34s
Epoch 44/120 | Train Loss: 0.3250 (acc. 85.95%) | Val Loss: 0.4087 (acc. 81.50%)
| Time: 34s
Epoch 45/120 | Train Loss: 0.3379 (acc. 85.60%) | Val Loss: 0.4383 (acc. 79.83%)
| Time: 34s
Epoch 46/120 | Train Loss: 0.3135 (acc. 86.95%) | Val Loss: 0.4213 (acc. 82.33%)
| Time: 34s
Epoch 47/120 | Train Loss: 0.3069 (acc. 86.90%) | Val Loss: 0.4665 (acc. 80.67%)
| Time: 34s
Epoch 48/120 | Train Loss: 0.3151 (acc. 86.00%) | Val Loss: 0.4264 (acc. 80.67%)
| Time: 34s
Epoch 49/120 | Train Loss: 0.3276 (acc. 86.00%) | Val Loss: 0.3953 (acc. 83.17%)
| Time: 34s
Epoch 50/120 | Train Loss: 0.3132 (acc. 86.35%) | Val Loss: 0.4257 (acc. 82.50%)
| Time: 34s
Epoch 51/120 | Train Loss: 0.3066 (acc. 87.05%) | Val Loss: 0.5127 (acc. 78.83%)
| Time: 34s
Epoch 52/120 | Train Loss: 0.3112 (acc. 86.45%) | Val Loss: 0.5312 (acc. 78.50%)
| Time: 34s
Epoch 53/120 | Train Loss: 0.3156 (acc. 86.65%) | Val Loss: 0.3942 (acc. 83.50%)

```

```

| Time: 34s
Epoch 54/120 | Train Loss: 0.3116 (acc. 86.35%) | Val Loss: 0.4176 (acc. 81.83%)
| Time: 34s
Epoch 55/120 | Train Loss: 0.2975 (acc. 87.35%) | Val Loss: 0.4386 (acc. 83.50%)
| Time: 34s
Epoch 56/120 | Train Loss: 0.3050 (acc. 86.00%) | Val Loss: 0.7170 (acc. 74.00%)
| Time: 34s
Epoch 57/120 | Train Loss: 0.3050 (acc. 87.30%) | Val Loss: 0.4180 (acc. 80.67%)
| Time: 34s
Epoch 58/120 | Train Loss: 0.3050 (acc. 87.45%) | Val Loss: 0.4370 (acc. 83.00%)
| Time: 34s
Epoch 59/120 | Train Loss: 0.3016 (acc. 87.30%) | Val Loss: 0.3755 (acc. 83.50%)
| Time: 34s
Epoch 60/120 | Train Loss: 0.2823 (acc. 87.10%) | Val Loss: 0.4339 (acc. 82.50%)
| Time: 34s
Epoch 61/120 | Train Loss: 0.2761 (acc. 88.35%) | Val Loss: 0.4827 (acc. 80.83%)
| Time: 34s
Epoch 62/120 | Train Loss: 0.2846 (acc. 87.85%) | Val Loss: 0.4043 (acc. 84.00%)
| Time: 34s
Epoch 63/120 | Train Loss: 0.3026 (acc. 86.60%) | Val Loss: 0.4255 (acc. 81.83%)
| Time: 34s
Epoch 64/120 | Train Loss: 0.2699 (acc. 88.85%) | Val Loss: 0.5582 (acc. 81.83%)
| Time: 34s
Epoch 65/120 | Train Loss: 0.2812 (acc. 87.60%) | Val Loss: 0.4267 (acc. 84.67%)
| Time: 34s
Epoch 66/120 | Train Loss: 0.2652 (acc. 89.10%) | Val Loss: 0.4230 (acc. 83.00%)
| Time: 34s
Epoch 67/120 | Train Loss: 0.2534 (acc. 89.50%) | Val Loss: 0.4107 (acc. 84.50%)
| Time: 34s
Epoch 68/120 | Train Loss: 0.2712 (acc. 88.05%) | Val Loss: 0.3898 (acc. 83.83%)
| Time: 34s
Epoch 69/120 | Train Loss: 0.2681 (acc. 88.85%) | Val Loss: 0.4008 (acc. 83.00%)
| Time: 34s
Epoch 70/120 | Train Loss: 0.2444 (acc. 89.45%) | Val Loss: 0.3871 (acc. 85.33%)
| Time: 34s
Epoch 71/120 | Train Loss: 0.2469 (acc. 89.15%) | Val Loss: 0.4005 (acc. 83.00%)
| Time: 34s
Epoch 72/120 | Train Loss: 0.2669 (acc. 89.15%) | Val Loss: 0.3852 (acc. 82.83%)
| Time: 34s
Epoch 73/120 | Train Loss: 0.2582 (acc. 88.95%) | Val Loss: 0.3484 (acc. 87.33%)
| Time: 34s
Epoch 74/120 | Train Loss: 0.2702 (acc. 89.15%) | Val Loss: 0.3432 (acc. 86.67%)
| Time: 34s
Epoch 75/120 | Train Loss: 0.2332 (acc. 90.90%) | Val Loss: 0.4229 (acc. 84.33%)
| Time: 34s
Epoch 76/120 | Train Loss: 0.2543 (acc. 89.50%) | Val Loss: 0.4300 (acc. 84.33%)
| Time: 34s
Epoch 77/120 | Train Loss: 0.2333 (acc. 90.00%) | Val Loss: 0.4691 (acc. 83.50%)
| Time: 34s
Epoch 78/120 | Train Loss: 0.2250 (acc. 90.50%) | Val Loss: 0.4165 (acc. 84.67%)
| Time: 34s
Epoch 79/120 | Train Loss: 0.2584 (acc. 87.75%) | Val Loss: 0.4136 (acc. 84.50%)
| Time: 34s
Epoch 80/120 | Train Loss: 0.2141 (acc. 90.85%) | Val Loss: 0.4059 (acc. 86.00%)
| Time: 34s
Epoch 81/120 | Train Loss: 0.2380 (acc. 89.85%) | Val Loss: 0.3663 (acc. 85.33%)
| Time: 34s
Epoch 82/120 | Train Loss: 0.2045 (acc. 91.65%) | Val Loss: 0.3784 (acc. 86.50%)
| Time: 34s
Epoch 83/120 | Train Loss: 0.2251 (acc. 90.80%) | Val Loss: 0.4138 (acc. 85.33%)
| Time: 34s
Epoch 84/120 | Train Loss: 0.2103 (acc. 91.55%) | Val Loss: 0.4051 (acc. 85.67%)
| Time: 34s
Epoch 85/120 | Train Loss: 0.2098 (acc. 90.80%) | Val Loss: 0.4096 (acc. 86.00%)
| Time: 34s
Epoch 86/120 | Train Loss: 0.2307 (acc. 90.75%) | Val Loss: 0.5740 (acc. 78.33%)
| Time: 34s
Epoch 87/120 | Train Loss: 0.2109 (acc. 91.40%) | Val Loss: 0.4140 (acc. 86.00%)

```

```

| Time: 34s
Epoch 88/120 | Train Loss: 0.2360 (acc. 90.40%) | Val Loss: 0.6556 (acc. 79.67%)
| Time: 34s
Epoch 89/120 | Train Loss: 0.2013 (acc. 91.50%) | Val Loss: 0.3833 (acc. 87.17%)
| Time: 34s
Epoch 90/120 | Train Loss: 0.1938 (acc. 92.55%) | Val Loss: 0.3877 (acc. 85.00%)
| Time: 34s
Epoch 91/120 | Train Loss: 0.1927 (acc. 92.95%) | Val Loss: 0.4278 (acc. 85.17%)
| Time: 34s
Epoch 92/120 | Train Loss: 0.2265 (acc. 91.05%) | Val Loss: 0.4150 (acc. 84.83%)
| Time: 34s
Epoch 93/120 | Train Loss: 0.2065 (acc. 91.95%) | Val Loss: 0.3894 (acc. 85.17%)
| Time: 34s
Epoch 94/120 | Train Loss: 0.1946 (acc. 91.65%) | Val Loss: 0.3629 (acc. 86.83%)
| Time: 34s
Epoch 95/120 | Train Loss: 0.1932 (acc. 92.35%) | Val Loss: 0.8695 (acc. 73.67%)
| Time: 34s
Epoch 96/120 | Train Loss: 0.2053 (acc. 92.25%) | Val Loss: 0.4317 (acc. 85.50%)
| Time: 34s
Epoch 97/120 | Train Loss: 0.1840 (acc. 93.00%) | Val Loss: 0.4594 (acc. 85.00%)
| Time: 34s
Epoch 98/120 | Train Loss: 0.2060 (acc. 91.90%) | Val Loss: 0.4642 (acc. 82.50%)
| Time: 34s
Epoch 99/120 | Train Loss: 0.1801 (acc. 92.45%) | Val Loss: 0.3906 (acc. 85.50%)
| Time: 34s
Epoch 100/120 | Train Loss: 0.1802 (acc. 93.25%) | Val Loss: 0.4055 (acc.
86.67%) | Time: 34s
Epoch 101/120 | Train Loss: 0.1609 (acc. 93.85%) | Val Loss: 0.4797 (acc.
84.50%) | Time: 34s
Epoch 102/120 | Train Loss: 0.1848 (acc. 93.05%) | Val Loss: 0.5259 (acc.
83.67%) | Time: 34s
Epoch 103/120 | Train Loss: 0.1772 (acc. 92.55%) | Val Loss: 0.6819 (acc.
79.17%) | Time: 34s
Epoch 104/120 | Train Loss: 0.1614 (acc. 93.15%) | Val Loss: 0.3821 (acc.
86.83%) | Time: 34s
Epoch 105/120 | Train Loss: 0.1660 (acc. 93.20%) | Val Loss: 0.4707 (acc.
85.33%) | Time: 34s
Epoch 106/120 | Train Loss: 0.1755 (acc. 93.05%) | Val Loss: 0.4738 (acc.
86.00%) | Time: 34s
Epoch 107/120 | Train Loss: 0.1603 (acc. 93.75%) | Val Loss: 0.4198 (acc.
86.17%) | Time: 34s
Epoch 108/120 | Train Loss: 0.1622 (acc. 93.65%) | Val Loss: 0.4181 (acc.
85.83%) | Time: 34s
Epoch 109/120 | Train Loss: 0.1887 (acc. 91.50%) | Val Loss: 0.7485 (acc.
81.00%) | Time: 34s
Epoch 110/120 | Train Loss: 0.1603 (acc. 94.00%) | Val Loss: 0.4439 (acc.
85.50%) | Time: 34s
Epoch 111/120 | Train Loss: 0.1324 (acc. 94.55%) | Val Loss: 0.3805 (acc.
87.83%) | Time: 34s
Epoch 112/120 | Train Loss: 0.1569 (acc. 93.70%) | Val Loss: 0.4204 (acc.
87.00%) | Time: 34s
Epoch 113/120 | Train Loss: 0.1614 (acc. 92.95%) | Val Loss: 0.4518 (acc.
85.67%) | Time: 34s
Epoch 114/120 | Train Loss: 0.1423 (acc. 94.70%) | Val Loss: 0.5452 (acc.
84.83%) | Time: 34s
Epoch 115/120 | Train Loss: 0.1704 (acc. 93.15%) | Val Loss: 0.4222 (acc.
86.17%) | Time: 34s
Epoch 116/120 | Train Loss: 0.1626 (acc. 92.75%) | Val Loss: 0.3715 (acc.
87.50%) | Time: 34s
Epoch 117/120 | Train Loss: 0.1605 (acc. 93.40%) | Val Loss: 0.4975 (acc.
85.00%) | Time: 34s
Epoch 118/120 | Train Loss: 0.1431 (acc. 93.75%) | Val Loss: 0.7164 (acc.
81.00%) | Time: 34s
Epoch 119/120 | Train Loss: 0.1447 (acc. 93.85%) | Val Loss: 0.4568 (acc.
86.17%) | Time: 34s
Epoch 120/120 | Train Loss: 0.1490 (acc. 94.25%) | Val Loss: 0.3999 (acc.
87.67%) | Time: 34s
Training Time: 4112s

```

```

Experiment: reg_2
Epoch 1/120 | Train Loss: 0.6836 (acc. 59.30%) | Val Loss: 0.6645 (acc. 60.33%)
| Time: 34s
Epoch 2/120 | Train Loss: 0.6343 (acc. 63.45%) | Val Loss: 0.7115 (acc. 60.00%)
| Time: 34s
Epoch 3/120 | Train Loss: 0.6051 (acc. 65.35%) | Val Loss: 0.6021 (acc. 67.83%)
| Time: 34s
Epoch 4/120 | Train Loss: 0.5782 (acc. 69.55%) | Val Loss: 0.6000 (acc. 68.50%)
| Time: 34s
Epoch 5/120 | Train Loss: 0.5681 (acc. 70.35%) | Val Loss: 0.6545 (acc. 61.83%)
| Time: 34s
Epoch 6/120 | Train Loss: 0.5626 (acc. 70.25%) | Val Loss: 0.6416 (acc. 65.83%)
| Time: 34s
Epoch 7/120 | Train Loss: 0.5540 (acc. 70.65%) | Val Loss: 0.5522 (acc. 71.83%)
| Time: 34s
Epoch 8/120 | Train Loss: 0.5263 (acc. 72.70%) | Val Loss: 0.5408 (acc. 73.67%)
| Time: 34s
Epoch 9/120 | Train Loss: 0.5294 (acc. 74.80%) | Val Loss: 0.5783 (acc. 66.67%)
| Time: 34s
Epoch 10/120 | Train Loss: 0.5449 (acc. 73.15%) | Val Loss: 0.5389 (acc. 72.67%)
| Time: 34s
Epoch 11/120 | Train Loss: 0.5085 (acc. 73.80%) | Val Loss: 0.5492 (acc. 72.50%)
| Time: 34s
Epoch 12/120 | Train Loss: 0.5174 (acc. 75.65%) | Val Loss: 0.5291 (acc. 74.17%)
| Time: 34s
Epoch 13/120 | Train Loss: 0.5021 (acc. 75.25%) | Val Loss: 0.5505 (acc. 72.83%)
| Time: 34s
Epoch 14/120 | Train Loss: 0.4755 (acc. 77.80%) | Val Loss: 0.5586 (acc. 73.17%)
| Time: 34s
Epoch 15/120 | Train Loss: 0.5140 (acc. 75.05%) | Val Loss: 0.5026 (acc. 75.67%)
| Time: 34s
Epoch 16/120 | Train Loss: 0.4761 (acc. 77.00%) | Val Loss: 0.5039 (acc. 75.33%)
| Time: 34s
Epoch 17/120 | Train Loss: 0.4528 (acc. 79.15%) | Val Loss: 0.5285 (acc. 75.00%)
| Time: 34s
Epoch 18/120 | Train Loss: 0.4641 (acc. 78.05%) | Val Loss: 0.4713 (acc. 77.33%)
| Time: 34s
Epoch 19/120 | Train Loss: 0.4549 (acc. 78.20%) | Val Loss: 0.6147 (acc. 72.17%)
| Time: 34s
Epoch 20/120 | Train Loss: 0.4364 (acc. 79.45%) | Val Loss: 0.5405 (acc. 77.00%)
| Time: 34s
Epoch 21/120 | Train Loss: 0.4396 (acc. 80.25%) | Val Loss: 0.5171 (acc. 76.33%)
| Time: 34s
Epoch 22/120 | Train Loss: 0.4259 (acc. 81.00%) | Val Loss: 0.4854 (acc. 76.17%)
| Time: 34s
Epoch 23/120 | Train Loss: 0.4458 (acc. 80.75%) | Val Loss: 0.4847 (acc. 75.67%)
| Time: 34s
Epoch 24/120 | Train Loss: 0.4250 (acc. 80.05%) | Val Loss: 0.4432 (acc. 79.17%)
| Time: 34s
Epoch 25/120 | Train Loss: 0.4174 (acc. 80.80%) | Val Loss: 0.5482 (acc. 72.50%)
| Time: 34s
Epoch 26/120 | Train Loss: 0.3966 (acc. 83.20%) | Val Loss: 0.4327 (acc. 79.83%)
| Time: 34s
Epoch 27/120 | Train Loss: 0.3544 (acc. 84.80%) | Val Loss: 0.4410 (acc. 79.50%)
| Time: 34s
Epoch 28/120 | Train Loss: 0.3779 (acc. 83.60%) | Val Loss: 0.5682 (acc. 78.00%)
| Time: 34s
Epoch 29/120 | Train Loss: 0.3680 (acc. 83.10%) | Val Loss: 0.4246 (acc. 80.00%)
| Time: 34s
Epoch 30/120 | Train Loss: 0.3695 (acc. 83.45%) | Val Loss: 0.4102 (acc. 81.83%)
| Time: 34s
Epoch 31/120 | Train Loss: 0.3606 (acc. 83.85%) | Val Loss: 0.4239 (acc. 79.50%)
| Time: 34s
Epoch 32/120 | Train Loss: 0.3457 (acc. 85.00%) | Val Loss: 0.4096 (acc. 81.83%)
| Time: 34s
Epoch 33/120 | Train Loss: 0.3729 (acc. 83.15%) | Val Loss: 0.4049 (acc. 81.67%)
| Time: 34s

```

```

Epoch 34/120 | Train Loss: 0.3425 (acc. 85.30%) | Val Loss: 0.4355 (acc. 78.67%)
| Time: 34s
Epoch 35/120 | Train Loss: 0.3354 (acc. 85.30%) | Val Loss: 0.4183 (acc. 79.17%)
| Time: 34s
Epoch 36/120 | Train Loss: 0.3577 (acc. 85.05%) | Val Loss: 0.4828 (acc. 80.33%)
| Time: 34s
Epoch 37/120 | Train Loss: 0.3456 (acc. 84.60%) | Val Loss: 0.3931 (acc. 82.50%)
| Time: 34s
Epoch 38/120 | Train Loss: 0.3322 (acc. 85.55%) | Val Loss: 0.3965 (acc. 83.17%)
| Time: 34s
Epoch 39/120 | Train Loss: 0.3058 (acc. 86.40%) | Val Loss: 0.4111 (acc. 82.83%)
| Time: 35s
Epoch 40/120 | Train Loss: 0.3404 (acc. 85.35%) | Val Loss: 0.4212 (acc. 79.83%)
| Time: 34s
Epoch 41/120 | Train Loss: 0.3401 (acc. 85.35%) | Val Loss: 0.4509 (acc. 80.83%)
| Time: 34s
Epoch 42/120 | Train Loss: 0.2948 (acc. 87.25%) | Val Loss: 0.3992 (acc. 82.67%)
| Time: 34s
Epoch 43/120 | Train Loss: 0.3022 (acc. 86.25%) | Val Loss: 0.3992 (acc. 83.00%)
| Time: 34s
Epoch 44/120 | Train Loss: 0.3162 (acc. 86.10%) | Val Loss: 0.4753 (acc. 81.50%)
| Time: 34s
Epoch 45/120 | Train Loss: 0.3307 (acc. 85.55%) | Val Loss: 0.4022 (acc. 82.17%)
| Time: 34s
Epoch 46/120 | Train Loss: 0.3134 (acc. 86.85%) | Val Loss: 0.3986 (acc. 83.33%)
| Time: 34s
Epoch 47/120 | Train Loss: 0.3016 (acc. 86.40%) | Val Loss: 0.5512 (acc. 76.83%)
| Time: 34s
Epoch 48/120 | Train Loss: 0.3121 (acc. 86.80%) | Val Loss: 0.4064 (acc. 82.17%)
| Time: 34s
Epoch 49/120 | Train Loss: 0.3283 (acc. 86.20%) | Val Loss: 0.4151 (acc. 81.83%)
| Time: 34s
Epoch 50/120 | Train Loss: 0.2865 (acc. 87.75%) | Val Loss: 0.4047 (acc. 80.00%)
| Time: 34s
Epoch 51/120 | Train Loss: 0.2695 (acc. 89.45%) | Val Loss: 0.5261 (acc. 78.67%)
| Time: 34s
Epoch 52/120 | Train Loss: 0.2730 (acc. 88.40%) | Val Loss: 0.3803 (acc. 83.00%)
| Time: 34s
Epoch 53/120 | Train Loss: 0.2533 (acc. 89.05%) | Val Loss: 0.3699 (acc. 83.00%)
| Time: 34s
Epoch 54/120 | Train Loss: 0.2447 (acc. 90.15%) | Val Loss: 0.3862 (acc. 82.50%)
| Time: 34s
Epoch 55/120 | Train Loss: 0.2711 (acc. 88.95%) | Val Loss: 0.4081 (acc. 82.17%)
| Time: 34s
Epoch 56/120 | Train Loss: 0.2641 (acc. 88.65%) | Val Loss: 0.3790 (acc. 83.67%)
| Time: 34s
Epoch 57/120 | Train Loss: 0.2395 (acc. 90.30%) | Val Loss: 0.3818 (acc. 84.17%)
| Time: 34s
Epoch 58/120 | Train Loss: 0.2488 (acc. 89.30%) | Val Loss: 0.4039 (acc. 83.33%)
| Time: 34s
Epoch 59/120 | Train Loss: 0.2345 (acc. 89.45%) | Val Loss: 0.4069 (acc. 83.33%)
| Time: 34s
Epoch 60/120 | Train Loss: 0.2545 (acc. 89.45%) | Val Loss: 0.3925 (acc. 81.67%)
| Time: 34s
Epoch 61/120 | Train Loss: 0.2547 (acc. 89.15%) | Val Loss: 0.3998 (acc. 82.00%)
| Time: 34s
Epoch 62/120 | Train Loss: 0.2528 (acc. 89.35%) | Val Loss: 0.3963 (acc. 83.33%)
| Time: 34s
Epoch 63/120 | Train Loss: 0.2303 (acc. 90.40%) | Val Loss: 0.3564 (acc. 84.17%)
| Time: 34s
Epoch 64/120 | Train Loss: 0.2378 (acc. 89.85%) | Val Loss: 0.3902 (acc. 83.17%)
| Time: 34s
Epoch 65/120 | Train Loss: 0.2459 (acc. 90.00%) | Val Loss: 0.4037 (acc. 83.33%)
| Time: 34s
Epoch 66/120 | Train Loss: 0.2132 (acc. 91.40%) | Val Loss: 0.4060 (acc. 82.50%)
| Time: 34s
Epoch 67/120 | Train Loss: 0.2265 (acc. 90.70%) | Val Loss: 0.4332 (acc. 82.83%)
| Time: 34s

```

```

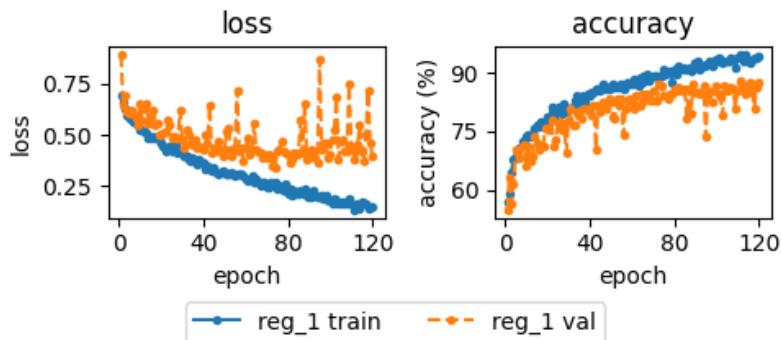
Epoch 68/120 | Train Loss: 0.2146 (acc. 91.25%) | Val Loss: 0.4085 (acc. 82.83%)
| Time: 34s
Epoch 69/120 | Train Loss: 0.2202 (acc. 90.65%) | Val Loss: 0.3930 (acc. 83.83%)
| Time: 34s
Epoch 70/120 | Train Loss: 0.2231 (acc. 90.00%) | Val Loss: 0.3705 (acc. 82.83%)
| Time: 34s
Epoch 71/120 | Train Loss: 0.2257 (acc. 90.90%) | Val Loss: 0.5548 (acc. 79.50%)
| Time: 34s
Epoch 72/120 | Train Loss: 0.2224 (acc. 90.10%) | Val Loss: 0.3889 (acc. 85.33%)
| Time: 34s
Epoch 73/120 | Train Loss: 0.2189 (acc. 90.40%) | Val Loss: 0.3737 (acc. 82.83%)
| Time: 34s
Epoch 74/120 | Train Loss: 0.2150 (acc. 91.35%) | Val Loss: 0.3786 (acc. 85.00%)
| Time: 34s
Epoch 75/120 | Train Loss: 0.1934 (acc. 91.40%) | Val Loss: 0.4055 (acc. 84.50%)
| Time: 34s
Epoch 76/120 | Train Loss: 0.2077 (acc. 91.25%) | Val Loss: 0.3716 (acc. 83.33%)
| Time: 34s
Epoch 77/120 | Train Loss: 0.1775 (acc. 92.90%) | Val Loss: 0.3672 (acc. 85.33%)
| Time: 34s
Epoch 78/120 | Train Loss: 0.1895 (acc. 91.70%) | Val Loss: 0.3786 (acc. 85.17%)
| Time: 34s
Epoch 79/120 | Train Loss: 0.1848 (acc. 92.00%) | Val Loss: 0.3830 (acc. 84.50%)
| Time: 34s
Epoch 80/120 | Train Loss: 0.1781 (acc. 92.90%) | Val Loss: 0.3858 (acc. 84.50%)
| Time: 34s
Epoch 81/120 | Train Loss: 0.1880 (acc. 92.50%) | Val Loss: 0.3819 (acc. 84.33%)
| Time: 34s
Epoch 82/120 | Train Loss: 0.1884 (acc. 92.70%) | Val Loss: 0.4351 (acc. 84.17%)
| Time: 34s
Epoch 83/120 | Train Loss: 0.2066 (acc. 92.20%) | Val Loss: 0.4312 (acc. 84.50%)
| Time: 34s
Epoch 84/120 | Train Loss: 0.1839 (acc. 92.30%) | Val Loss: 0.3898 (acc. 84.67%)
| Time: 34s
Epoch 85/120 | Train Loss: 0.1813 (acc. 92.65%) | Val Loss: 0.4003 (acc. 84.67%)
| Time: 34s
Epoch 86/120 | Train Loss: 0.1831 (acc. 92.30%) | Val Loss: 0.3869 (acc. 84.67%)
| Time: 34s
Epoch 87/120 | Train Loss: 0.1765 (acc. 93.15%) | Val Loss: 0.3765 (acc. 84.50%)
| Time: 34s
Epoch 88/120 | Train Loss: 0.1671 (acc. 93.00%) | Val Loss: 0.3776 (acc. 84.17%)
| Time: 34s
Epoch 89/120 | Train Loss: 0.1641 (acc. 93.55%) | Val Loss: 0.3761 (acc. 85.50%)
| Time: 34s
Epoch 90/120 | Train Loss: 0.1715 (acc. 92.65%) | Val Loss: 0.4062 (acc. 84.83%)
| Time: 34s
Epoch 91/120 | Train Loss: 0.1762 (acc. 92.65%) | Val Loss: 0.3752 (acc. 85.67%)
| Time: 34s
Epoch 92/120 | Train Loss: 0.1673 (acc. 93.35%) | Val Loss: 0.4252 (acc. 83.67%)
| Time: 34s
Epoch 93/120 | Train Loss: 0.1655 (acc. 93.70%) | Val Loss: 0.3844 (acc. 85.33%)
| Time: 34s
Epoch 94/120 | Train Loss: 0.1607 (acc. 93.30%) | Val Loss: 0.3873 (acc. 85.83%)
| Time: 34s
Epoch 95/120 | Train Loss: 0.1577 (acc. 93.45%) | Val Loss: 0.3750 (acc. 84.67%)
| Time: 34s
Epoch 96/120 | Train Loss: 0.1712 (acc. 93.15%) | Val Loss: 0.3911 (acc. 85.67%)
| Time: 34s
Epoch 97/120 | Train Loss: 0.1462 (acc. 94.35%) | Val Loss: 0.3730 (acc. 85.67%)
| Time: 34s
Epoch 98/120 | Train Loss: 0.1576 (acc. 93.35%) | Val Loss: 0.4075 (acc. 84.67%)
| Time: 34s
Epoch 99/120 | Train Loss: 0.1636 (acc. 93.60%) | Val Loss: 0.3994 (acc. 85.17%)
| Time: 34s
Epoch 100/120 | Train Loss: 0.1412 (acc. 94.25%) | Val Loss: 0.4108 (acc.
85.17%) | Time: 34s
Epoch 101/120 | Train Loss: 0.1632 (acc. 93.05%) | Val Loss: 0.3921 (acc.
86.17%) | Time: 34s

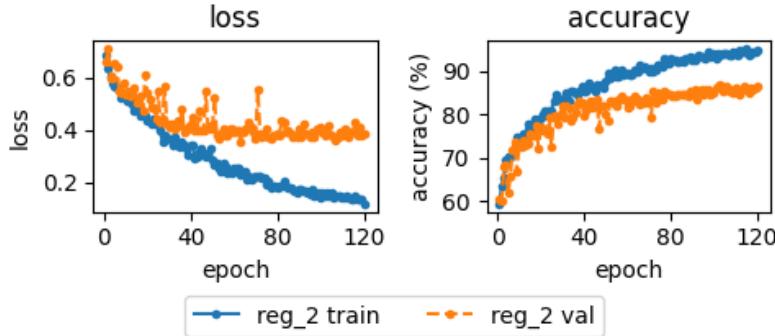
```

```

Epoch 102/120 | Train Loss: 0.1473 (acc. 94.10%) | Val Loss: 0.3816 (acc.
86.50%) | Time: 34s
Epoch 103/120 | Train Loss: 0.1616 (acc. 93.85%) | Val Loss: 0.3904 (acc.
86.83%) | Time: 34s
Epoch 104/120 | Train Loss: 0.1503 (acc. 94.05%) | Val Loss: 0.4195 (acc.
85.33%) | Time: 34s
Epoch 105/120 | Train Loss: 0.1607 (acc. 93.95%) | Val Loss: 0.3830 (acc.
86.17%) | Time: 34s
Epoch 106/120 | Train Loss: 0.1445 (acc. 94.15%) | Val Loss: 0.3717 (acc.
86.67%) | Time: 34s
Epoch 107/120 | Train Loss: 0.1506 (acc. 93.90%) | Val Loss: 0.3647 (acc.
85.50%) | Time: 34s
Epoch 108/120 | Train Loss: 0.1465 (acc. 93.80%) | Val Loss: 0.3742 (acc.
85.67%) | Time: 34s
Epoch 109/120 | Train Loss: 0.1480 (acc. 94.65%) | Val Loss: 0.3903 (acc.
85.17%) | Time: 34s
Epoch 110/120 | Train Loss: 0.1402 (acc. 93.95%) | Val Loss: 0.3808 (acc.
86.33%) | Time: 34s
Epoch 111/120 | Train Loss: 0.1420 (acc. 93.65%) | Val Loss: 0.4025 (acc.
85.33%) | Time: 34s
Epoch 112/120 | Train Loss: 0.1481 (acc. 94.80%) | Val Loss: 0.4009 (acc.
84.50%) | Time: 34s
Epoch 113/120 | Train Loss: 0.1451 (acc. 94.60%) | Val Loss: 0.3759 (acc.
85.83%) | Time: 34s
Epoch 114/120 | Train Loss: 0.1347 (acc. 94.50%) | Val Loss: 0.3902 (acc.
86.50%) | Time: 34s
Epoch 115/120 | Train Loss: 0.1350 (acc. 95.25%) | Val Loss: 0.3958 (acc.
85.83%) | Time: 34s
Epoch 116/120 | Train Loss: 0.1492 (acc. 93.60%) | Val Loss: 0.4291 (acc.
85.17%) | Time: 34s
Epoch 117/120 | Train Loss: 0.1390 (acc. 94.25%) | Val Loss: 0.3823 (acc.
86.33%) | Time: 35s
Epoch 118/120 | Train Loss: 0.1388 (acc. 94.50%) | Val Loss: 0.4105 (acc.
85.83%) | Time: 34s
Epoch 119/120 | Train Loss: 0.1387 (acc. 94.65%) | Val Loss: 0.3825 (acc.
86.17%) | Time: 34s
Epoch 120/120 | Train Loss: 0.1183 (acc. 94.95%) | Val Loss: 0.3888 (acc.
86.50%) | Time: 34s
Training Time: 4112s

```





0.9.1 Adding one more convolutional layer

```
[22]: net_config2 = {**default_net_config,
    "cv_layers": [
        {"out_channels": 64, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
        {"out_channels": 128, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
        {"out_channels": 256, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
        {"out_channels": 512, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
        {"out_channels": 512, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
    ],
    "fc_layers": [
        {"out_features": 512, "batch_norm": True, "dropout_rate": 0.4},
        {"out_features": 256, "batch_norm": True, "dropout_rate": 0.2},
    ],
}
configs2 = [
    {**default_config, "label": "reg_3", "net_config": net_config2},
    {**default_config, "label": "reg_4", "net_config": net_config2, "train_config": train_config1}
]
results = result_handler(configs2, device)
plot_scores(results)
```

```
Experiment: reg_3
Epoch 1/120 | Train Loss: 0.7206 (acc. 55.15%) | Val Loss: 0.7543 (acc. 56.83%)
| Time: 67s
Epoch 2/120 | Train Loss: 0.6802 (acc. 57.90%) | Val Loss: 0.7889 (acc. 58.50%)
| Time: 67s
Epoch 3/120 | Train Loss: 0.6636 (acc. 60.65%) | Val Loss: 0.6234 (acc. 63.00%)
| Time: 67s
Epoch 4/120 | Train Loss: 0.6367 (acc. 63.45%) | Val Loss: 0.5956 (acc. 68.83%)
| Time: 67s
Epoch 5/120 | Train Loss: 0.6149 (acc. 66.10%) | Val Loss: 0.6728 (acc. 61.50%)
| Time: 67s
Epoch 6/120 | Train Loss: 0.6022 (acc. 66.60%) | Val Loss: 0.6197 (acc. 65.00%)
| Time: 67s
Epoch 7/120 | Train Loss: 0.5728 (acc. 70.70%) | Val Loss: 0.5960 (acc. 68.67%)
| Time: 67s
Epoch 8/120 | Train Loss: 0.5657 (acc. 71.15%) | Val Loss: 0.7474 (acc. 61.00%)
| Time: 67s
```

```

Epoch 9/120 | Train Loss: 0.5561 (acc. 71.15%) | Val Loss: 0.5889 (acc. 70.33%)
| Time: 67s
Epoch 10/120 | Train Loss: 0.5503 (acc. 72.65%) | Val Loss: 0.5609 (acc. 71.17%)
| Time: 67s
Epoch 11/120 | Train Loss: 0.5269 (acc. 72.90%) | Val Loss: 0.5995 (acc. 64.33%)
| Time: 67s
Epoch 12/120 | Train Loss: 0.5106 (acc. 74.65%) | Val Loss: 0.5758 (acc. 72.50%)
| Time: 67s
Epoch 13/120 | Train Loss: 0.5094 (acc. 75.35%) | Val Loss: 0.5395 (acc. 74.83%)
| Time: 67s
Epoch 14/120 | Train Loss: 0.4990 (acc. 75.45%) | Val Loss: 0.5470 (acc. 74.00%)
| Time: 67s
Epoch 15/120 | Train Loss: 0.4923 (acc. 76.70%) | Val Loss: 0.6012 (acc. 68.67%)
| Time: 67s
Epoch 16/120 | Train Loss: 0.4936 (acc. 76.90%) | Val Loss: 0.4990 (acc. 74.83%)
| Time: 67s
Epoch 17/120 | Train Loss: 0.4631 (acc. 79.30%) | Val Loss: 0.5559 (acc. 72.17%)
| Time: 67s
Epoch 18/120 | Train Loss: 0.4567 (acc. 78.95%) | Val Loss: 0.5044 (acc. 73.50%)
| Time: 67s
Epoch 19/120 | Train Loss: 0.4417 (acc. 78.95%) | Val Loss: 0.5147 (acc. 75.83%)
| Time: 67s
Epoch 20/120 | Train Loss: 0.4621 (acc. 78.85%) | Val Loss: 0.4939 (acc. 78.17%)
| Time: 67s
Epoch 21/120 | Train Loss: 0.4443 (acc. 79.30%) | Val Loss: 0.5131 (acc. 75.17%)
| Time: 67s
Epoch 22/120 | Train Loss: 0.4118 (acc. 80.95%) | Val Loss: 0.6186 (acc. 73.50%)
| Time: 67s
Epoch 23/120 | Train Loss: 0.4778 (acc. 76.40%) | Val Loss: 0.4806 (acc. 78.00%)
| Time: 67s
Epoch 24/120 | Train Loss: 0.4142 (acc. 80.80%) | Val Loss: 0.4869 (acc. 77.17%)
| Time: 67s
Epoch 25/120 | Train Loss: 0.4148 (acc. 80.40%) | Val Loss: 0.4357 (acc. 78.00%)
| Time: 67s
Epoch 26/120 | Train Loss: 0.4125 (acc. 81.50%) | Val Loss: 0.4279 (acc. 81.17%)
| Time: 67s
Epoch 27/120 | Train Loss: 0.3962 (acc. 81.20%) | Val Loss: 0.4663 (acc. 79.33%)
| Time: 67s
Epoch 28/120 | Train Loss: 0.3772 (acc. 84.10%) | Val Loss: 0.4881 (acc. 78.50%)
| Time: 67s
Epoch 29/120 | Train Loss: 0.3432 (acc. 84.60%) | Val Loss: 0.4451 (acc. 81.00%)
| Time: 67s
Epoch 30/120 | Train Loss: 0.3646 (acc. 83.65%) | Val Loss: 0.4301 (acc. 81.17%)
| Time: 67s
Epoch 31/120 | Train Loss: 0.3487 (acc. 84.25%) | Val Loss: 0.5731 (acc. 76.00%)
| Time: 67s
Epoch 32/120 | Train Loss: 0.3363 (acc. 84.75%) | Val Loss: 0.5071 (acc. 76.17%)
| Time: 67s
Epoch 33/120 | Train Loss: 0.3626 (acc. 83.65%) | Val Loss: 0.3897 (acc. 82.50%)
| Time: 67s
Epoch 34/120 | Train Loss: 0.3289 (acc. 85.75%) | Val Loss: 0.3771 (acc. 86.00%)
| Time: 67s
Epoch 35/120 | Train Loss: 0.3029 (acc. 87.25%) | Val Loss: 0.4087 (acc. 84.00%)
| Time: 67s
Epoch 36/120 | Train Loss: 0.3089 (acc. 86.50%) | Val Loss: 0.3707 (acc. 84.00%)
| Time: 67s
Epoch 37/120 | Train Loss: 0.3221 (acc. 86.55%) | Val Loss: 0.4977 (acc. 80.00%)
| Time: 67s
Epoch 38/120 | Train Loss: 0.2892 (acc. 87.15%) | Val Loss: 0.3727 (acc. 85.17%)
| Time: 67s
Epoch 39/120 | Train Loss: 0.2993 (acc. 87.10%) | Val Loss: 0.4675 (acc. 81.17%)
| Time: 67s
Epoch 40/120 | Train Loss: 0.3062 (acc. 86.55%) | Val Loss: 0.3523 (acc. 84.83%)
| Time: 67s
Epoch 41/120 | Train Loss: 0.2927 (acc. 87.40%) | Val Loss: 0.3551 (acc. 83.83%)
| Time: 67s
Epoch 42/120 | Train Loss: 0.2845 (acc. 87.90%) | Val Loss: 0.3976 (acc. 83.50%)
| Time: 67s

```

```

Epoch 43/120 | Train Loss: 0.2866 (acc. 88.35%) | Val Loss: 0.6133 (acc. 78.50%)
| Time: 67s
Epoch 44/120 | Train Loss: 0.2843 (acc. 88.35%) | Val Loss: 0.3339 (acc. 85.50%)
| Time: 67s
Epoch 45/120 | Train Loss: 0.2659 (acc. 89.25%) | Val Loss: 0.3488 (acc. 84.67%)
| Time: 67s
Epoch 46/120 | Train Loss: 0.2761 (acc. 88.70%) | Val Loss: 0.3933 (acc. 86.33%)
| Time: 67s
Epoch 47/120 | Train Loss: 0.2852 (acc. 87.85%) | Val Loss: 0.3365 (acc. 84.83%)
| Time: 67s
Epoch 48/120 | Train Loss: 0.2639 (acc. 88.00%) | Val Loss: 0.3755 (acc. 84.00%)
| Time: 67s
Epoch 49/120 | Train Loss: 0.2497 (acc. 89.95%) | Val Loss: 0.7493 (acc. 73.17%)
| Time: 67s
Epoch 50/120 | Train Loss: 0.2600 (acc. 89.60%) | Val Loss: 0.2911 (acc. 88.33%)
| Time: 67s
Epoch 51/120 | Train Loss: 0.2528 (acc. 89.40%) | Val Loss: 0.3326 (acc. 86.00%)
| Time: 67s
Epoch 52/120 | Train Loss: 0.2187 (acc. 90.60%) | Val Loss: 0.4604 (acc. 82.83%)
| Time: 67s
Epoch 53/120 | Train Loss: 0.2296 (acc. 90.75%) | Val Loss: 0.3377 (acc. 86.00%)
| Time: 67s
Epoch 54/120 | Train Loss: 0.2347 (acc. 90.05%) | Val Loss: 0.4674 (acc. 82.50%)
| Time: 67s
Epoch 55/120 | Train Loss: 0.2162 (acc. 90.70%) | Val Loss: 0.3766 (acc. 85.00%)
| Time: 67s
Epoch 56/120 | Train Loss: 0.1940 (acc. 92.00%) | Val Loss: 0.3656 (acc. 86.33%)
| Time: 67s
Epoch 57/120 | Train Loss: 0.1849 (acc. 92.70%) | Val Loss: 0.3781 (acc. 85.67%)
| Time: 67s
Epoch 58/120 | Train Loss: 0.2236 (acc. 90.80%) | Val Loss: 0.7648 (acc. 75.83%)
| Time: 67s
Epoch 59/120 | Train Loss: 0.1955 (acc. 91.55%) | Val Loss: 0.4405 (acc. 84.00%)
| Time: 67s
Epoch 60/120 | Train Loss: 0.2079 (acc. 91.60%) | Val Loss: 0.3520 (acc. 87.00%)
| Time: 67s
Epoch 61/120 | Train Loss: 0.2027 (acc. 91.70%) | Val Loss: 0.3833 (acc. 86.17%)
| Time: 67s
Epoch 62/120 | Train Loss: 0.2129 (acc. 90.40%) | Val Loss: 0.3591 (acc. 86.00%)
| Time: 67s
Epoch 63/120 | Train Loss: 0.1757 (acc. 92.80%) | Val Loss: 0.3098 (acc. 88.33%)
| Time: 67s
Epoch 64/120 | Train Loss: 0.1723 (acc. 92.95%) | Val Loss: 0.3120 (acc. 87.50%)
| Time: 67s
Epoch 65/120 | Train Loss: 0.1906 (acc. 92.45%) | Val Loss: 0.3507 (acc. 86.83%)
| Time: 67s
Epoch 66/120 | Train Loss: 0.1713 (acc. 92.95%) | Val Loss: 0.3310 (acc. 87.17%)
| Time: 67s
Epoch 67/120 | Train Loss: 0.1631 (acc. 93.30%) | Val Loss: 0.2971 (acc. 89.00%)
| Time: 67s
Epoch 68/120 | Train Loss: 0.1673 (acc. 93.75%) | Val Loss: 0.3352 (acc. 88.67%)
| Time: 67s
Epoch 69/120 | Train Loss: 0.1557 (acc. 93.70%) | Val Loss: 0.4033 (acc. 87.17%)
| Time: 67s
Epoch 70/120 | Train Loss: 0.1711 (acc. 93.10%) | Val Loss: 0.3080 (acc. 89.67%)
| Time: 67s
Epoch 71/120 | Train Loss: 0.1537 (acc. 93.90%) | Val Loss: 0.3449 (acc. 86.83%)
| Time: 67s
Epoch 72/120 | Train Loss: 0.1642 (acc. 93.20%) | Val Loss: 0.3177 (acc. 88.00%)
| Time: 67s
Epoch 73/120 | Train Loss: 0.1374 (acc. 94.15%) | Val Loss: 0.2684 (acc. 88.83%)
| Time: 67s
Epoch 74/120 | Train Loss: 0.1266 (acc. 95.05%) | Val Loss: 0.2855 (acc. 88.83%)
| Time: 67s
Epoch 75/120 | Train Loss: 0.1609 (acc. 93.70%) | Val Loss: 0.3289 (acc. 89.33%)
| Time: 67s
Epoch 76/120 | Train Loss: 0.1678 (acc. 92.65%) | Val Loss: 0.2932 (acc. 88.17%)
| Time: 67s

```

```

Epoch 77/120 | Train Loss: 0.1437 (acc. 94.80%) | Val Loss: 0.4449 (acc. 85.17%)
| Time: 67s
Epoch 78/120 | Train Loss: 0.1824 (acc. 92.70%) | Val Loss: 0.3284 (acc. 87.00%)
| Time: 67s
Epoch 79/120 | Train Loss: 0.1366 (acc. 94.65%) | Val Loss: 0.3170 (acc. 87.33%)
| Time: 67s
Epoch 80/120 | Train Loss: 0.1377 (acc. 94.35%) | Val Loss: 0.3692 (acc. 87.50%)
| Time: 67s
Epoch 81/120 | Train Loss: 0.1326 (acc. 95.30%) | Val Loss: 0.2757 (acc. 87.67%)
| Time: 67s
Epoch 82/120 | Train Loss: 0.1139 (acc. 95.55%) | Val Loss: 0.3848 (acc. 87.17%)
| Time: 68s
Epoch 83/120 | Train Loss: 0.1520 (acc. 93.85%) | Val Loss: 0.3718 (acc. 86.00%)
| Time: 67s
Epoch 84/120 | Train Loss: 0.1272 (acc. 95.35%) | Val Loss: 0.3369 (acc. 87.00%)
| Time: 67s
Epoch 85/120 | Train Loss: 0.1166 (acc. 95.50%) | Val Loss: 0.2958 (acc. 89.67%)
| Time: 67s
Epoch 86/120 | Train Loss: 0.1233 (acc. 95.45%) | Val Loss: 0.3126 (acc. 88.50%)
| Time: 67s
Epoch 87/120 | Train Loss: 0.1655 (acc. 94.05%) | Val Loss: 0.3738 (acc. 87.67%)
| Time: 67s
Epoch 88/120 | Train Loss: 0.1296 (acc. 94.65%) | Val Loss: 0.2697 (acc. 90.00%)
| Time: 67s
Epoch 89/120 | Train Loss: 0.1159 (acc. 95.75%) | Val Loss: 0.5481 (acc. 85.00%)
| Time: 67s
Epoch 90/120 | Train Loss: 0.1388 (acc. 94.20%) | Val Loss: 0.4467 (acc. 87.17%)
| Time: 67s
Epoch 91/120 | Train Loss: 0.1342 (acc. 94.90%) | Val Loss: 0.2801 (acc. 90.83%)
| Time: 67s
Epoch 92/120 | Train Loss: 0.1434 (acc. 94.15%) | Val Loss: 0.3517 (acc. 88.50%)
| Time: 67s
Epoch 93/120 | Train Loss: 0.1240 (acc. 95.05%) | Val Loss: 0.2855 (acc. 89.83%)
| Time: 67s
Epoch 94/120 | Train Loss: 0.0916 (acc. 96.35%) | Val Loss: 0.3218 (acc. 90.33%)
| Time: 67s
Epoch 95/120 | Train Loss: 0.0975 (acc. 96.00%) | Val Loss: 0.5325 (acc. 85.17%)
| Time: 67s
Epoch 96/120 | Train Loss: 0.1017 (acc. 96.30%) | Val Loss: 0.3723 (acc. 88.00%)
| Time: 67s
Epoch 97/120 | Train Loss: 0.1237 (acc. 94.30%) | Val Loss: 0.2634 (acc. 91.00%)
| Time: 67s
Epoch 98/120 | Train Loss: 0.1012 (acc. 95.90%) | Val Loss: 0.3041 (acc. 90.33%)
| Time: 67s
Epoch 99/120 | Train Loss: 0.1016 (acc. 96.25%) | Val Loss: 0.3535 (acc. 88.33%)
| Time: 67s
Epoch 100/120 | Train Loss: 0.0990 (acc. 95.95%) | Val Loss: 0.2743 (acc.
91.17%) | Time: 67s
Epoch 101/120 | Train Loss: 0.1105 (acc. 95.70%) | Val Loss: 0.2681 (acc.
90.33%) | Time: 67s
Epoch 102/120 | Train Loss: 0.1064 (acc. 96.00%) | Val Loss: 0.4660 (acc.
88.00%) | Time: 67s
Epoch 103/120 | Train Loss: 0.1173 (acc. 95.05%) | Val Loss: 0.2934 (acc.
89.83%) | Time: 67s
Epoch 104/120 | Train Loss: 0.1039 (acc. 95.80%) | Val Loss: 0.2386 (acc.
91.50%) | Time: 67s
Epoch 105/120 | Train Loss: 0.1126 (acc. 96.50%) | Val Loss: 0.3063 (acc.
90.83%) | Time: 67s
Epoch 106/120 | Train Loss: 0.1080 (acc. 96.30%) | Val Loss: 0.3350 (acc.
89.67%) | Time: 67s
Epoch 107/120 | Train Loss: 0.0932 (acc. 96.20%) | Val Loss: 0.2827 (acc.
90.33%) | Time: 67s
Epoch 108/120 | Train Loss: 0.0998 (acc. 96.50%) | Val Loss: 0.3170 (acc.
89.67%) | Time: 67s
Epoch 109/120 | Train Loss: 0.1282 (acc. 95.30%) | Val Loss: 0.3020 (acc.
89.50%) | Time: 67s
Epoch 110/120 | Train Loss: 0.0889 (acc. 96.60%) | Val Loss: 0.4448 (acc.
85.83%) | Time: 67s

```

```

Epoch 111/120 | Train Loss: 0.0856 (acc. 96.65%) | Val Loss: 0.2558 (acc.
90.00%) | Time: 67s
Epoch 112/120 | Train Loss: 0.0883 (acc. 96.85%) | Val Loss: 0.2908 (acc.
90.17%) | Time: 67s
Epoch 113/120 | Train Loss: 0.0900 (acc. 96.55%) | Val Loss: 0.2865 (acc.
90.67%) | Time: 67s
Epoch 114/120 | Train Loss: 0.0760 (acc. 97.10%) | Val Loss: 0.2750 (acc.
91.33%) | Time: 67s
Epoch 115/120 | Train Loss: 0.0827 (acc. 96.75%) | Val Loss: 0.2618 (acc.
92.17%) | Time: 67s
Epoch 116/120 | Train Loss: 0.1015 (acc. 96.25%) | Val Loss: 0.2775 (acc.
90.33%) | Time: 67s
Epoch 117/120 | Train Loss: 0.0798 (acc. 96.95%) | Val Loss: 0.3634 (acc.
88.17%) | Time: 67s
Epoch 118/120 | Train Loss: 0.0842 (acc. 96.95%) | Val Loss: 0.2854 (acc.
90.67%) | Time: 67s
Epoch 119/120 | Train Loss: 0.0669 (acc. 97.25%) | Val Loss: 0.2490 (acc.
91.83%) | Time: 67s
Epoch 120/120 | Train Loss: 0.0740 (acc. 97.15%) | Val Loss: 0.3117 (acc.
91.67%) | Time: 67s
Training Time: 8018s

Experiment: reg_4
Epoch 1/120 | Train Loss: 0.7047 (acc. 55.85%) | Val Loss: 0.6922 (acc. 62.00%)
| Time: 67s
Epoch 2/120 | Train Loss: 0.6393 (acc. 63.35%) | Val Loss: 0.6482 (acc. 59.83%)
| Time: 67s
Epoch 3/120 | Train Loss: 0.6206 (acc. 66.35%) | Val Loss: 0.6200 (acc. 64.17%)
| Time: 67s
Epoch 4/120 | Train Loss: 0.5876 (acc. 68.50%) | Val Loss: 0.6980 (acc. 62.17%)
| Time: 67s
Epoch 5/120 | Train Loss: 0.5801 (acc. 69.85%) | Val Loss: 0.6019 (acc. 66.17%)
| Time: 67s
Epoch 6/120 | Train Loss: 0.5693 (acc. 68.80%) | Val Loss: 0.5931 (acc. 68.17%)
| Time: 67s
Epoch 7/120 | Train Loss: 0.5562 (acc. 72.05%) | Val Loss: 0.5676 (acc. 69.17%)
| Time: 67s
Epoch 8/120 | Train Loss: 0.5508 (acc. 73.50%) | Val Loss: 0.6502 (acc. 65.50%)
| Time: 67s
Epoch 9/120 | Train Loss: 0.5197 (acc. 73.90%) | Val Loss: 0.5756 (acc. 72.00%)
| Time: 67s
Epoch 10/120 | Train Loss: 0.5229 (acc. 74.70%) | Val Loss: 0.6711 (acc. 62.50%)
| Time: 67s
Epoch 11/120 | Train Loss: 0.5215 (acc. 75.45%) | Val Loss: 0.5439 (acc. 74.50%)
| Time: 67s
Epoch 12/120 | Train Loss: 0.4906 (acc. 76.55%) | Val Loss: 0.5044 (acc. 76.67%)
| Time: 67s
Epoch 13/120 | Train Loss: 0.4835 (acc. 77.25%) | Val Loss: 0.4873 (acc. 75.83%)
| Time: 67s
Epoch 14/120 | Train Loss: 0.4614 (acc. 79.60%) | Val Loss: 0.6430 (acc. 69.33%)
| Time: 67s
Epoch 15/120 | Train Loss: 0.4500 (acc. 79.40%) | Val Loss: 0.6143 (acc. 72.00%)
| Time: 67s
Epoch 16/120 | Train Loss: 0.4687 (acc. 78.00%) | Val Loss: 0.5107 (acc. 77.00%)
| Time: 67s
Epoch 17/120 | Train Loss: 0.4408 (acc. 79.45%) | Val Loss: 0.5049 (acc. 75.50%)
| Time: 67s
Epoch 18/120 | Train Loss: 0.4183 (acc. 81.80%) | Val Loss: 0.4865 (acc. 76.50%)
| Time: 67s
Epoch 19/120 | Train Loss: 0.4158 (acc. 81.90%) | Val Loss: 0.4882 (acc. 78.50%)
| Time: 67s
Epoch 20/120 | Train Loss: 0.3973 (acc. 81.45%) | Val Loss: 0.5123 (acc. 77.67%)
| Time: 67s
Epoch 21/120 | Train Loss: 0.3712 (acc. 83.45%) | Val Loss: 0.4798 (acc. 78.33%)
| Time: 67s
Epoch 22/120 | Train Loss: 0.4021 (acc. 83.05%) | Val Loss: 0.5558 (acc. 74.17%)
| Time: 67s
Epoch 23/120 | Train Loss: 0.3716 (acc. 83.20%) | Val Loss: 0.4363 (acc. 81.33%)

```

```

| Time: 67s
Epoch 24/120 | Train Loss: 0.3868 (acc. 81.95%) | Val Loss: 0.7918 (acc. 69.50%)
| Time: 67s
Epoch 25/120 | Train Loss: 0.3843 (acc. 82.60%) | Val Loss: 0.4914 (acc. 79.17%)
| Time: 67s
Epoch 26/120 | Train Loss: 0.3303 (acc. 86.25%) | Val Loss: 0.3908 (acc. 81.83%)
| Time: 67s
Epoch 27/120 | Train Loss: 0.3031 (acc. 86.35%) | Val Loss: 0.4620 (acc. 81.67%)
| Time: 67s
Epoch 28/120 | Train Loss: 0.2944 (acc. 87.65%) | Val Loss: 0.4229 (acc. 80.83%)
| Time: 67s
Epoch 29/120 | Train Loss: 0.3225 (acc. 85.80%) | Val Loss: 0.3870 (acc. 83.83%)
| Time: 67s
Epoch 30/120 | Train Loss: 0.3118 (acc. 87.05%) | Val Loss: 0.3783 (acc. 83.17%)
| Time: 67s
Epoch 31/120 | Train Loss: 0.2989 (acc. 86.45%) | Val Loss: 0.3883 (acc. 84.67%)
| Time: 67s
Epoch 32/120 | Train Loss: 0.2765 (acc. 88.50%) | Val Loss: 0.3599 (acc. 85.33%)
| Time: 67s
Epoch 33/120 | Train Loss: 0.2972 (acc. 87.10%) | Val Loss: 0.3706 (acc. 84.83%)
| Time: 67s
Epoch 34/120 | Train Loss: 0.2757 (acc. 89.10%) | Val Loss: 0.3927 (acc. 83.50%)
| Time: 68s
Epoch 35/120 | Train Loss: 0.2514 (acc. 90.20%) | Val Loss: 0.3298 (acc. 86.00%)
| Time: 68s
Epoch 36/120 | Train Loss: 0.2860 (acc. 87.55%) | Val Loss: 0.3622 (acc. 85.00%)
| Time: 67s
Epoch 37/120 | Train Loss: 0.2438 (acc. 89.35%) | Val Loss: 0.3580 (acc. 85.50%)
| Time: 67s
Epoch 38/120 | Train Loss: 0.2286 (acc. 90.60%) | Val Loss: 0.3856 (acc. 84.00%)
| Time: 67s
Epoch 39/120 | Train Loss: 0.2383 (acc. 90.10%) | Val Loss: 0.3534 (acc. 85.33%)
| Time: 67s
Epoch 40/120 | Train Loss: 0.2089 (acc. 91.50%) | Val Loss: 0.3663 (acc. 85.00%)
| Time: 68s
Epoch 41/120 | Train Loss: 0.2168 (acc. 91.05%) | Val Loss: 0.3586 (acc. 84.17%)
| Time: 67s
Epoch 42/120 | Train Loss: 0.2216 (acc. 90.95%) | Val Loss: 0.4370 (acc. 84.33%)
| Time: 70s
Epoch 43/120 | Train Loss: 0.2208 (acc. 90.60%) | Val Loss: 0.3776 (acc. 85.50%)
| Time: 67s
Epoch 44/120 | Train Loss: 0.2285 (acc. 90.15%) | Val Loss: 0.3789 (acc. 84.67%)
| Time: 67s
Epoch 45/120 | Train Loss: 0.2159 (acc. 90.85%) | Val Loss: 0.4297 (acc. 86.50%)
| Time: 67s
Epoch 46/120 | Train Loss: 0.2089 (acc. 91.30%) | Val Loss: 0.3336 (acc. 86.67%)
| Time: 67s
Epoch 47/120 | Train Loss: 0.1955 (acc. 92.10%) | Val Loss: 0.5513 (acc. 81.50%)
| Time: 67s
Epoch 48/120 | Train Loss: 0.2087 (acc. 91.45%) | Val Loss: 0.3444 (acc. 86.33%)
| Time: 67s
Epoch 49/120 | Train Loss: 0.2035 (acc. 91.45%) | Val Loss: 0.4692 (acc. 85.33%)
| Time: 67s
Epoch 50/120 | Train Loss: 0.1823 (acc. 91.95%) | Val Loss: 0.3462 (acc. 87.33%)
| Time: 67s
Epoch 51/120 | Train Loss: 0.1732 (acc. 93.45%) | Val Loss: 0.3008 (acc. 89.00%)
| Time: 67s
Epoch 52/120 | Train Loss: 0.1742 (acc. 92.75%) | Val Loss: 0.3278 (acc. 87.33%)
| Time: 67s
Epoch 53/120 | Train Loss: 0.1615 (acc. 93.55%) | Val Loss: 0.3221 (acc. 88.00%)
| Time: 67s
Epoch 54/120 | Train Loss: 0.1429 (acc. 94.00%) | Val Loss: 0.3693 (acc. 87.00%)
| Time: 67s
Epoch 55/120 | Train Loss: 0.1290 (acc. 94.85%) | Val Loss: 0.4544 (acc. 86.17%)
| Time: 67s
Epoch 56/120 | Train Loss: 0.1425 (acc. 94.20%) | Val Loss: 0.3564 (acc. 87.17%)
| Time: 67s
Epoch 57/120 | Train Loss: 0.1544 (acc. 93.95%) | Val Loss: 0.3282 (acc. 88.33%)

```

```

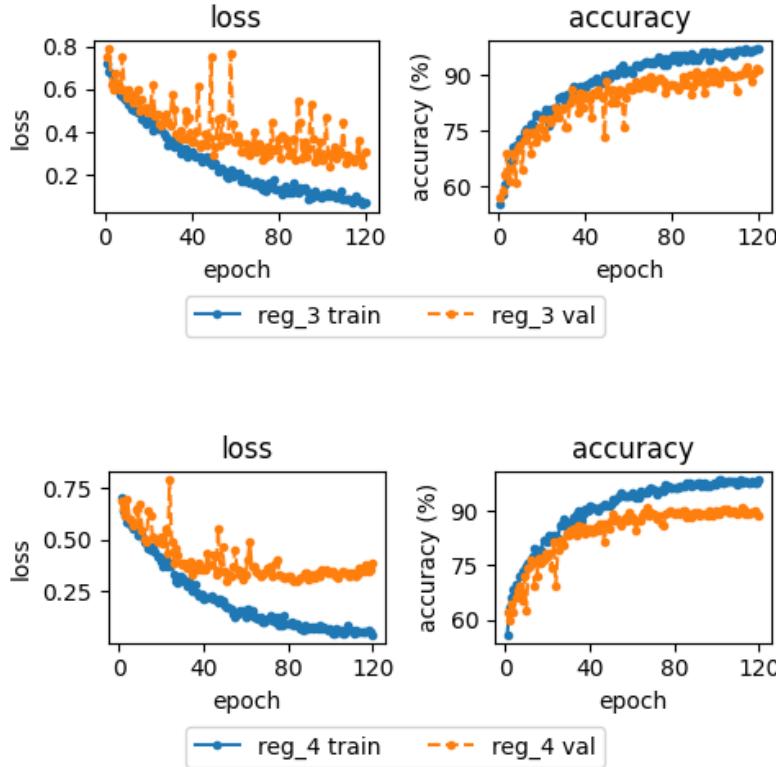
| Time: 67s
Epoch 58/120 | Train Loss: 0.1483 (acc. 94.25%) | Val Loss: 0.3485 (acc. 87.33%)
| Time: 67s
Epoch 59/120 | Train Loss: 0.1574 (acc. 93.95%) | Val Loss: 0.3070 (acc. 89.50%)
| Time: 67s
Epoch 60/120 | Train Loss: 0.1253 (acc. 95.10%) | Val Loss: 0.3293 (acc. 88.17%)
| Time: 67s
Epoch 61/120 | Train Loss: 0.1670 (acc. 94.75%) | Val Loss: 0.4046 (acc. 87.50%)
| Time: 67s
Epoch 62/120 | Train Loss: 0.1581 (acc. 93.65%) | Val Loss: 0.4881 (acc. 84.83%)
| Time: 67s
Epoch 63/120 | Train Loss: 0.1563 (acc. 93.85%) | Val Loss: 0.3700 (acc. 87.50%)
| Time: 67s
Epoch 64/120 | Train Loss: 0.1228 (acc. 95.65%) | Val Loss: 0.3689 (acc. 87.33%)
| Time: 67s
Epoch 65/120 | Train Loss: 0.1254 (acc. 95.05%) | Val Loss: 0.3460 (acc. 88.83%)
| Time: 67s
Epoch 66/120 | Train Loss: 0.1189 (acc. 95.20%) | Val Loss: 0.3387 (acc. 89.50%)
| Time: 67s
Epoch 67/120 | Train Loss: 0.1289 (acc. 95.00%) | Val Loss: 0.3249 (acc. 91.00%)
| Time: 67s
Epoch 68/120 | Train Loss: 0.1008 (acc. 96.10%) | Val Loss: 0.3334 (acc. 88.83%)
| Time: 67s
Epoch 69/120 | Train Loss: 0.0984 (acc. 96.20%) | Val Loss: 0.3412 (acc. 89.50%)
| Time: 67s
Epoch 70/120 | Train Loss: 0.1160 (acc. 95.95%) | Val Loss: 0.3617 (acc. 89.17%)
| Time: 67s
Epoch 71/120 | Train Loss: 0.1178 (acc. 95.65%) | Val Loss: 0.3561 (acc. 88.67%)
| Time: 67s
Epoch 72/120 | Train Loss: 0.1307 (acc. 94.50%) | Val Loss: 0.3677 (acc. 87.83%)
| Time: 67s
Epoch 73/120 | Train Loss: 0.1076 (acc. 95.65%) | Val Loss: 0.3262 (acc. 88.83%)
| Time: 67s
Epoch 74/120 | Train Loss: 0.1048 (acc. 96.10%) | Val Loss: 0.3870 (acc. 86.33%)
| Time: 67s
Epoch 75/120 | Train Loss: 0.1219 (acc. 95.70%) | Val Loss: 0.4031 (acc. 86.17%)
| Time: 67s
Epoch 76/120 | Train Loss: 0.0832 (acc. 97.15%) | Val Loss: 0.3331 (acc. 89.33%)
| Time: 67s
Epoch 77/120 | Train Loss: 0.0879 (acc. 96.85%) | Val Loss: 0.3198 (acc. 89.50%)
| Time: 67s
Epoch 78/120 | Train Loss: 0.1347 (acc. 96.40%) | Val Loss: 0.3182 (acc. 90.00%)
| Time: 67s
Epoch 79/120 | Train Loss: 0.0964 (acc. 96.50%) | Val Loss: 0.3149 (acc. 89.50%)
| Time: 67s
Epoch 80/120 | Train Loss: 0.0950 (acc. 96.25%) | Val Loss: 0.3244 (acc. 89.33%)
| Time: 67s
Epoch 81/120 | Train Loss: 0.0960 (acc. 96.45%) | Val Loss: 0.3273 (acc. 89.67%)
| Time: 67s
Epoch 82/120 | Train Loss: 0.0992 (acc. 96.45%) | Val Loss: 0.3052 (acc. 89.50%)
| Time: 67s
Epoch 83/120 | Train Loss: 0.0868 (acc. 97.05%) | Val Loss: 0.3015 (acc. 89.00%)
| Time: 67s
Epoch 84/120 | Train Loss: 0.0902 (acc. 96.65%) | Val Loss: 0.2983 (acc. 90.00%)
| Time: 67s
Epoch 85/120 | Train Loss: 0.0824 (acc. 96.75%) | Val Loss: 0.3060 (acc. 90.00%)
| Time: 67s
Epoch 86/120 | Train Loss: 0.0683 (acc. 97.70%) | Val Loss: 0.3179 (acc. 88.17%)
| Time: 67s
Epoch 87/120 | Train Loss: 0.0651 (acc. 97.65%) | Val Loss: 0.3229 (acc. 89.17%)
| Time: 67s
Epoch 88/120 | Train Loss: 0.0832 (acc. 96.70%) | Val Loss: 0.3387 (acc. 88.33%)
| Time: 67s
Epoch 89/120 | Train Loss: 0.0677 (acc. 97.15%) | Val Loss: 0.3406 (acc. 89.17%)
| Time: 67s
Epoch 90/120 | Train Loss: 0.0668 (acc. 97.25%) | Val Loss: 0.3052 (acc. 89.67%)
| Time: 67s
Epoch 91/120 | Train Loss: 0.0771 (acc. 97.55%) | Val Loss: 0.3154 (acc. 89.83%)

```

```

| Time: 67s
Epoch 92/120 | Train Loss: 0.0714 (acc. 97.60%) | Val Loss: 0.3325 (acc. 88.83%)
| Time: 67s
Epoch 93/120 | Train Loss: 0.0774 (acc. 97.05%) | Val Loss: 0.3195 (acc. 89.17%)
| Time: 67s
Epoch 94/120 | Train Loss: 0.0665 (acc. 97.60%) | Val Loss: 0.3369 (acc. 89.33%)
| Time: 67s
Epoch 95/120 | Train Loss: 0.0829 (acc. 96.95%) | Val Loss: 0.3272 (acc. 89.33%)
| Time: 67s
Epoch 96/120 | Train Loss: 0.0704 (acc. 97.35%) | Val Loss: 0.3520 (acc. 88.83%)
| Time: 67s
Epoch 97/120 | Train Loss: 0.0814 (acc. 96.75%) | Val Loss: 0.3474 (acc. 90.00%)
| Time: 67s
Epoch 98/120 | Train Loss: 0.0725 (acc. 97.10%) | Val Loss: 0.3524 (acc. 88.67%)
| Time: 67s
Epoch 99/120 | Train Loss: 0.0598 (acc. 97.70%) | Val Loss: 0.3396 (acc. 89.50%)
| Time: 67s
Epoch 100/120 | Train Loss: 0.0665 (acc. 97.20%) | Val Loss: 0.3376 (acc.
90.17%) | Time: 67s
Epoch 101/120 | Train Loss: 0.0456 (acc. 98.50%) | Val Loss: 0.3500 (acc.
89.00%) | Time: 67s
Epoch 102/120 | Train Loss: 0.0472 (acc. 98.50%) | Val Loss: 0.3313 (acc.
90.17%) | Time: 67s
Epoch 103/120 | Train Loss: 0.0674 (acc. 97.70%) | Val Loss: 0.3263 (acc.
90.17%) | Time: 67s
Epoch 104/120 | Train Loss: 0.0554 (acc. 97.95%) | Val Loss: 0.3132 (acc.
90.67%) | Time: 67s
Epoch 105/120 | Train Loss: 0.0589 (acc. 98.00%) | Val Loss: 0.3411 (acc.
90.67%) | Time: 67s
Epoch 106/120 | Train Loss: 0.0456 (acc. 98.05%) | Val Loss: 0.3470 (acc.
89.83%) | Time: 67s
Epoch 107/120 | Train Loss: 0.0433 (acc. 98.30%) | Val Loss: 0.3356 (acc.
89.67%) | Time: 67s
Epoch 108/120 | Train Loss: 0.0856 (acc. 98.25%) | Val Loss: 0.3408 (acc.
90.00%) | Time: 67s
Epoch 109/120 | Train Loss: 0.0578 (acc. 97.95%) | Val Loss: 0.3249 (acc.
89.50%) | Time: 68s
Epoch 110/120 | Train Loss: 0.0625 (acc. 97.45%) | Val Loss: 0.3357 (acc.
89.33%) | Time: 68s
Epoch 111/120 | Train Loss: 0.0448 (acc. 98.60%) | Val Loss: 0.3400 (acc.
89.50%) | Time: 68s
Epoch 112/120 | Train Loss: 0.0578 (acc. 97.95%) | Val Loss: 0.3215 (acc.
90.83%) | Time: 67s
Epoch 113/120 | Train Loss: 0.0525 (acc. 98.15%) | Val Loss: 0.3348 (acc.
90.50%) | Time: 68s
Epoch 114/120 | Train Loss: 0.0510 (acc. 98.00%) | Val Loss: 0.3671 (acc.
89.17%) | Time: 67s
Epoch 115/120 | Train Loss: 0.0532 (acc. 97.95%) | Val Loss: 0.3569 (acc.
89.33%) | Time: 68s
Epoch 116/120 | Train Loss: 0.0564 (acc. 97.85%) | Val Loss: 0.3501 (acc.
89.50%) | Time: 68s
Epoch 117/120 | Train Loss: 0.0544 (acc. 98.00%) | Val Loss: 0.3818 (acc.
89.67%) | Time: 67s
Epoch 118/120 | Train Loss: 0.0606 (acc. 97.85%) | Val Loss: 0.3494 (acc.
90.17%) | Time: 67s
Epoch 119/120 | Train Loss: 0.0563 (acc. 97.65%) | Val Loss: 0.3548 (acc.
89.33%) | Time: 68s
Epoch 120/120 | Train Loss: 0.0384 (acc. 98.65%) | Val Loss: 0.3839 (acc.
88.67%) | Time: 67s
Training Time: 8039s

```



0.10 Predict

```
[27]: test_image = "data/test/cats/cat.1306.jpg"
net_config2 = {**default_net_config,
    "cv_layers": [
        {"out_channels": 64, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
        {"out_channels": 128, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
        {"out_channels": 256, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
        {"out_channels": 512, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
        {"out_channels": 512, "kernel_size": 3, "stride": 1, "padding": 1, "max_pool": 2, "max_pool_stride": 2, "batch_norm": True},
    ],
    "fc_layers": [
        {"out_features": 512, "batch_norm": True, "dropout_rate": 0.4},
        {"out_features": 256, "batch_norm": True, "dropout_rate": 0.2},
    ],
}
model = ConvolutionalNetwork(net_config2)
predict(model, device, "models/reg_4.pth", "results/reg_4.csv")
plot_individual_feature_maps(model, device, test_image, "results/feature_maps_reg_4.png")
```

Total Correctly Classified: 366 | Total Misclassified: 34 | Accuracy: 91.5%
 Class-wise Correctly Classified Counts:

```

- Class cat: 175
- Class dog: 191
Class-wise Misclassified Counts:
- Class cat: 25
- Class dog: 9
Cat Accuracy: 87.5% | Dog Accuracy: 95.5%

```



Original Image













0.11 Transfer Learning

```
[28]: num_classes = default_config["net_config"]["num_classes"]

model_alexnet = alexnet(weights=AlexNet_Weights.DEFAULT)

# Freeze all parameters/layers except last one
for name, param in model_alexnet.named_parameters():
    if("bn" not in name):
        param.requires_grad = False

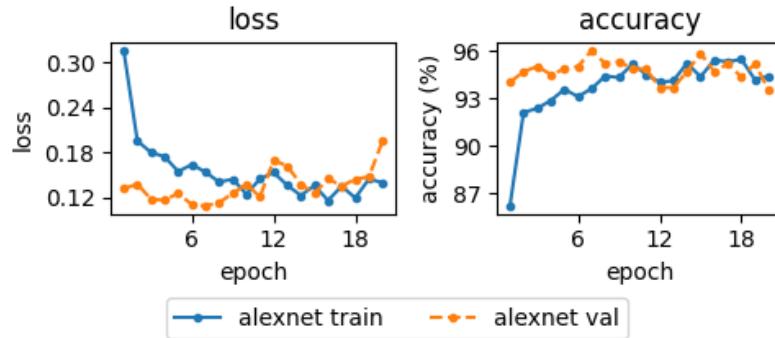
# Change last layer to output 2 classes
model_alexnet.classifier[6] = nn.Linear(4096, num_classes)

config_alexnet = {**default_config, "label": "alexnet", "n_epochs": 20}

results = {}
results["alexnet"] = train_model(model_alexnet, device, config_alexnet)
plot_scores(results)
```

Experiment: alexnet

Epoch 1/20 | Train Loss: 0.3162 (acc. 86.20%) | Val Loss: 0.1319 (acc. 94.00%) | Time: 43s
Epoch 2/20 | Train Loss: 0.1956 (acc. 92.10%) | Val Loss: 0.1380 (acc. 94.67%) | Time: 38s
Epoch 3/20 | Train Loss: 0.1803 (acc. 92.35%) | Val Loss: 0.1180 (acc. 95.00%) | Time: 37s
Epoch 4/20 | Train Loss: 0.1746 (acc. 92.85%) | Val Loss: 0.1164 (acc. 94.50%) | Time: 38s
Epoch 5/20 | Train Loss: 0.1548 (acc. 93.55%) | Val Loss: 0.1260 (acc. 94.83%) | Time: 37s
Epoch 6/20 | Train Loss: 0.1642 (acc. 93.10%) | Val Loss: 0.1105 (acc. 95.00%) | Time: 36s
Epoch 7/20 | Train Loss: 0.1540 (acc. 93.60%) | Val Loss: 0.1089 (acc. 96.00%) | Time: 36s
Epoch 8/20 | Train Loss: 0.1409 (acc. 94.40%) | Val Loss: 0.1133 (acc. 95.17%) | Time: 35s
Epoch 9/20 | Train Loss: 0.1446 (acc. 94.30%) | Val Loss: 0.1256 (acc. 95.33%) | Time: 36s
Epoch 10/20 | Train Loss: 0.1248 (acc. 95.15%) | Val Loss: 0.1380 (acc. 94.83%) | Time: 35s
Epoch 11/20 | Train Loss: 0.1448 (acc. 94.50%) | Val Loss: 0.1210 (acc. 94.83%) | Time: 36s
Epoch 12/20 | Train Loss: 0.1536 (acc. 94.00%) | Val Loss: 0.1698 (acc. 93.67%) | Time: 35s
Epoch 13/20 | Train Loss: 0.1365 (acc. 94.10%) | Val Loss: 0.1622 (acc. 93.67%) | Time: 35s
Epoch 14/20 | Train Loss: 0.1229 (acc. 95.20%) | Val Loss: 0.1379 (acc. 94.67%) | Time: 35s
Epoch 15/20 | Train Loss: 0.1361 (acc. 94.35%) | Val Loss: 0.1254 (acc. 95.83%) | Time: 36s
Epoch 16/20 | Train Loss: 0.1163 (acc. 95.40%) | Val Loss: 0.1454 (acc. 94.67%) | Time: 36s
Epoch 17/20 | Train Loss: 0.1351 (acc. 95.30%) | Val Loss: 0.1353 (acc. 95.17%) | Time: 40s
Epoch 18/20 | Train Loss: 0.1196 (acc. 95.45%) | Val Loss: 0.1446 (acc. 94.33%) | Time: 35s
Epoch 19/20 | Train Loss: 0.1448 (acc. 94.15%) | Val Loss: 0.1477 (acc. 95.17%) | Time: 33s
Epoch 20/20 | Train Loss: 0.1400 (acc. 94.40%) | Val Loss: 0.1953 (acc. 93.50%) | Time: 33s
Training Time: 724s



```
[26]: model = AlexNet(num_classes=num_classes)
predict(model, device, "models/alexnet.pth", "results/alexnet.csv")
plot_individual_feature_maps(model, device, test_image, "results/feature_maps_alexnet.png")
```

Total Correctly Classified: 374 | Total Misclassified: 26 | Accuracy: 93.5%

Class-wise Correctly Classified Counts:

- Class cat: 191
- Class dog: 183

Class-wise Misclassified Counts:

- Class cat: 9
- Class dog: 17

Cat Accuracy: 95.5% | Dog Accuracy: 91.5%



Original Image



