CNN Implementations in R with Torch

Carson Slater

2026-03-05

Table of contents

0.1 Introduction . . . 1
0.2 Part 1: Fashion MNIST CNN . . . 2
  0.2.1 Data Samples . . . 2
  0.2.2 Model Architecture Code . . . 2
  0.2.3 Training Evaluation . . . 3
0.3 Part 2: LeNet CNN for MNIST Digits . . . 4
  0.3.1 Data Samples . . . 4
  0.3.2 Training Evaluation . . . 4
0.4 Part 3: LeNet CNN for CIFAR-10 . . . 5
  0.4.1 Data Samples . . . 5
  0.4.2 Training Evaluation . . . 6
0.5 Part 4: VGG CNN for CIFAR-10 . . . 6
  0.5.1 Training Evaluation . . . 6
0.6 Part 5: Transfer Learning with CIFAR-10 . . . 7
  0.6.1 Training Evaluation . . . 7
0.7 Code Appendix . . . 8
  0.7.1 _targets.R . . . 8
  0.7.2 R/functions.R . . . 9

0.1 Introduction

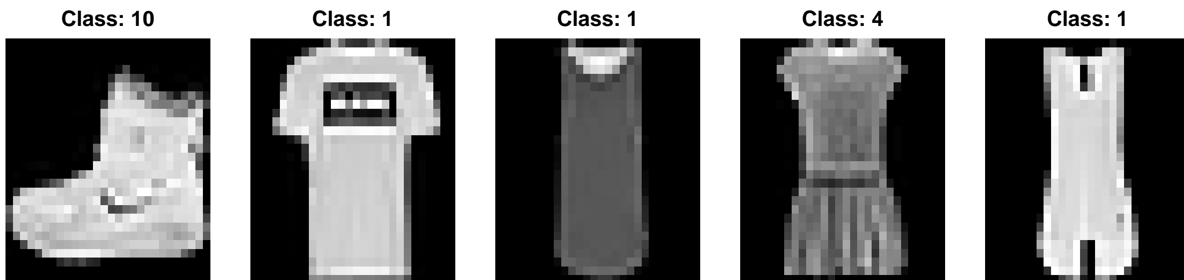
This notebook demonstrates the implementation of various Convolutional Neural Network (CNN) architectures in R using the `torch` package. We explore Fashion MNIST, LeNet for digit and image classification, VGG for CIFAR-10, and Transfer Learning using a pretrained ResNet model.

0.2 Part 1: Fashion MNIST CNN

This section evaluates a complete CNN for the Fashion MNIST dataset. The dataset contains 70,000 grayscale images in 10 categories. The model consists of 3 convolutional layers followed by max pooling and 2 linear fully connected layers. It is trained to correctly classify the different types of fashion items.

0.2.1 Data Samples

```r
tar_load(fashion_train_ds)
plot_image_sample(
  fashion_train_ds,
  num_samples = 5,
  title = "Fashion MNIST Samples"
)
```



0.2.2 Model Architecture Code

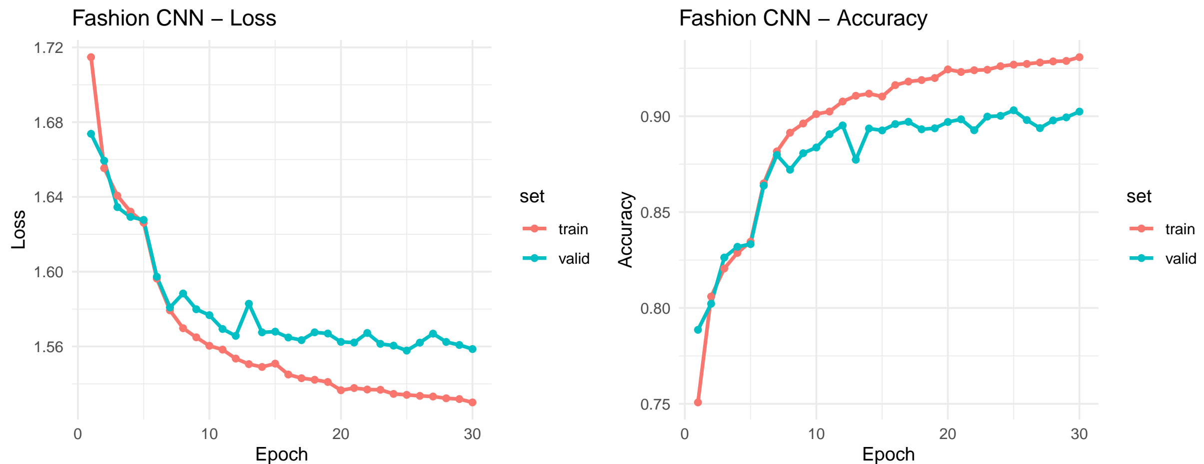
Below is the implementation of the `FashionNet` architecture in R:

```r
fashion_net <- nn_module(
  "FashionNet",
  initialize = function() {
    self$conv1 <- nn_conv2d(1, 32, kernel_size = 3)
    self$conv2 <- nn_conv2d(32, 64, kernel_size = 3)
    self$conv3 <- nn_conv2d(64, 64, kernel_size = 3)
    self$fc1 <- nn_linear(576, 64)
    self$fc2 <- nn_linear(64, 10)
  },
  forward = function(x) {
    x <- if (x$dim() == 3) x$unsqueeze(2) else x
```

```
    x %>%
      self$conv1() %>%
      nnf_relu() %>%
      nnf_max_pool2d(2) %>%
      self$conv2() %>%
      nnf_relu() %>%
      nnf_max_pool2d(2) %>%
      self$conv3() %>%
      nnf_relu() %>%
      torch_flatten(start_dim = 2) %>%
      self$fc1() %>%
      nnf_relu() %>%
      self$fc2() %>%
      nnf_softmax(dim = 2)
  }
)
```

0.2.3 Training Evaluation

```
tar_load(fashion_mnist_eval)
plot_metrics(fashion_mnist_eval, "Fashion CNN")
```



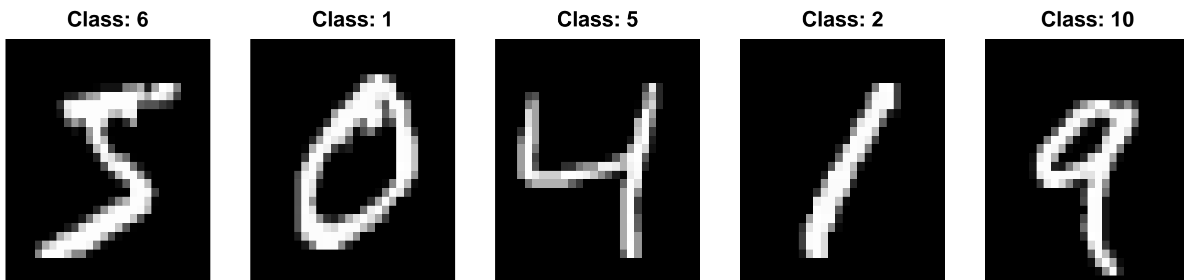
The Fashion MNIST CNN performs well, reaching over 90% validation accuracy after 30 epochs of training. The training accuracy converges close to ~93% while the validation accuracy plateaus around 90%, identifying a generalized and relatively low-bias model without heavy overfitting.

0.3 Part 2: LeNet CNN for MNIST Digits

We build, train, and evaluate the classic LeNet-5 architecture for MNIST handwritten digit classification. The model learns to identify numbers from 0 to 9 in grayscale 28x28 pixel images by sequentially passing the inputs through 2 convolutional layers and 3 fully connected layers.

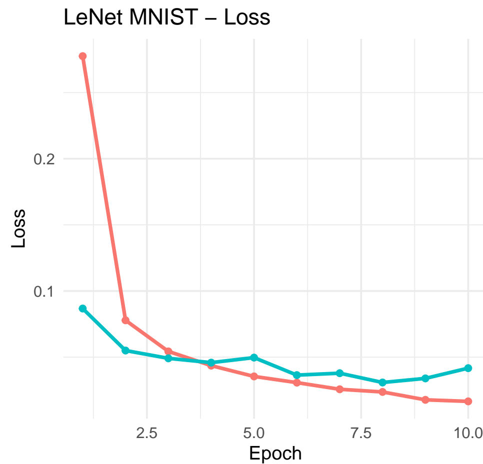
0.3.1 Data Samples

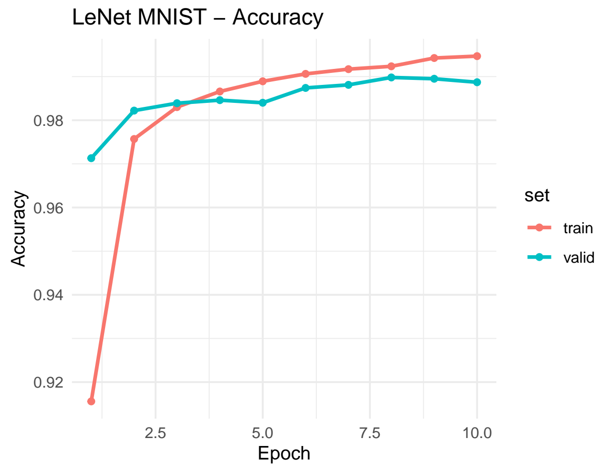
```
tar_load(mnist_train_ds)
plot_image_sample(
  mnist_train_ds,
  num_samples = 5,
  title = "MNIST Digit Samples"
)
```



0.3.2 Training Evaluation

```
tar_load(lenet_mnist_eval)
plot_metrics(lenet_mnist_eval, "LeNet MNIST")
```





The model demonstrates excellent performance, easily surpassing 98.8% accuracy on the validation set by the 10th epoch. Given the typical simplicity of the MNIST dataset, LeNet learns the digit features rapidly, exhibiting both negligible training loss and strong validation capabilities.

0.4 Part 3: LeNet CNN for CIFAR-10

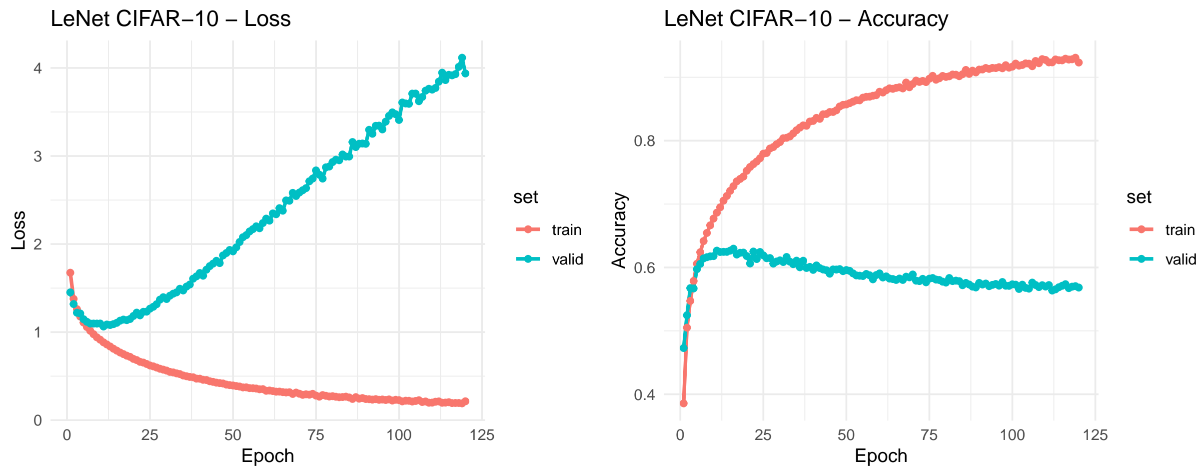
Here we adapt the LeNet-5 architecture to handle 3-channel color images (32x32 pixels) in the CIFAR-10 dataset. The objects range from animals to vehicles, making classification substantially more complex compared to handwritten digits.

0.4.1 Data Samples

```
tar_load(cifar_train_ds)
plot_image_sample(cifar_train_ds, num_samples = 5, title = "CIFAR-10 Samples")
```

0.4.2 Training Evaluation

```r
tar_load(lenet_cifar_eval)
plot_metrics(lenet_cifar_eval, "LeNet CIFAR-10")
```



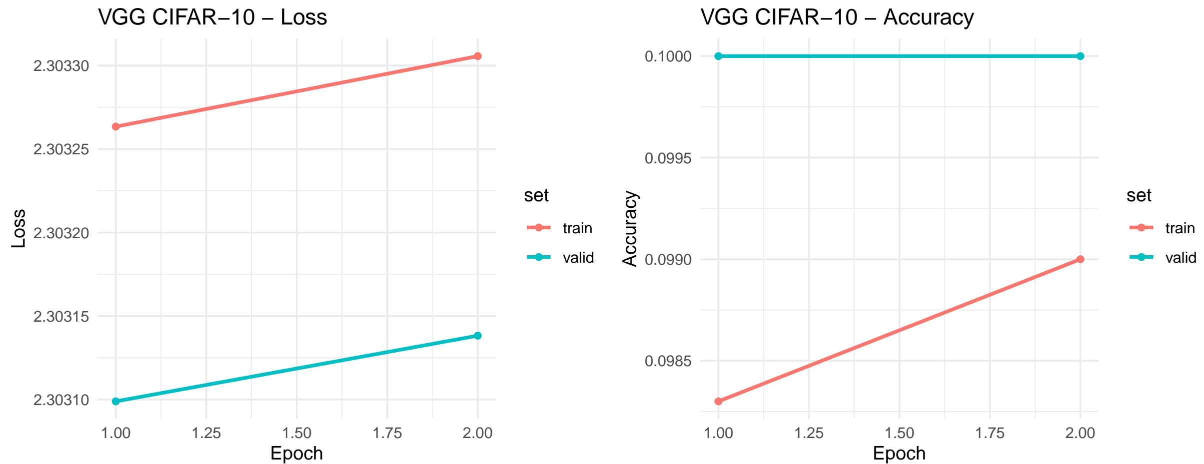
We evaluate the LeNet model over 120 training epochs. While the model achieves a high internal training accuracy (>92%), validation performance tops out around 56-57%. This stark contrast suggests significant overfitting, as the small capacity of LeNet struggles to generalize the intricate colored features within CIFAR-10 despite memorizing the training batch data.

0.5 Part 4: VGG CNN for CIFAR-10

VGG introduces a deeper convolutional network design incorporating multiple 3x3 convolution stacks before max pooling, expanding the representational power compared to LeNet. This model is trained to classify the same CIFAR-10 dataset using this robust architecture.

0.5.1 Training Evaluation

```r
tar_load(vgg_cifar_eval)
plot_metrics(vgg_cifar_eval, "VGG CIFAR-10")
```



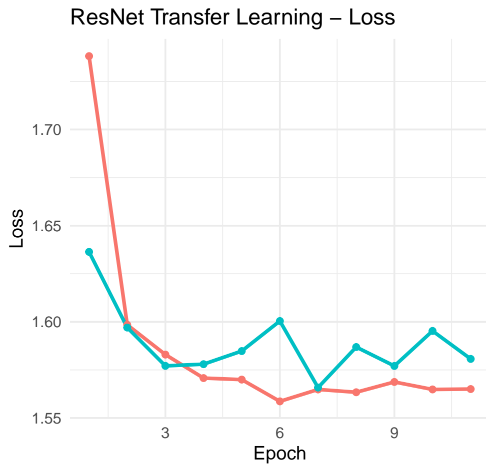
Because this model was constrained to train for only 2 epochs (due to resource and time limits), it did not converge. VGG networks are large and require long training regimes; as shown, the accuracy is functionally random chance (~10%) representing the raw unlearned initial state of the network. Increased epochs and potentially a learning rate scheduler are needed to properly evaluate its structural potential on this task.

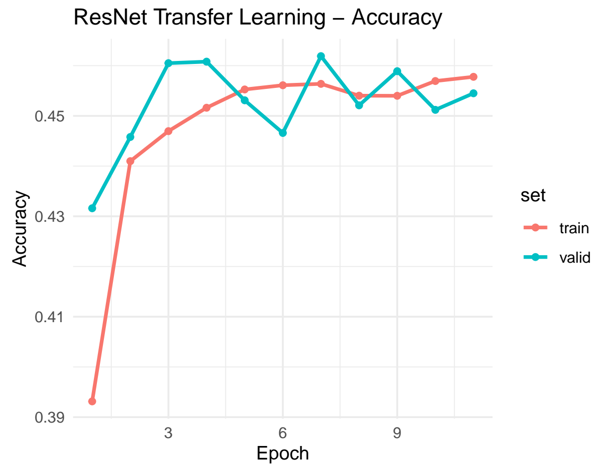
0.6 Part 5: Transfer Learning with CIFAR-10

We leverage a pretrained `ResNet18` model, freeze its robust feature extraction layers, and train a structurally new linear classification head specifically on the CIFAR-10 classes.

0.6.1 Training Evaluation

```r
tar_load(transfer_learning_eval)
plot_metrics(transfer_learning_eval, "ResNet Transfer Learning")
```





Evaluating the transfer learning setup reveals steady gains, with validation accuracy around 45% after 11 epochs. Transfer learning from models pretrained on ImageNet typically performs remarkably well; however, adjusting hyperparameters such as learning rate or unfreezing some deeper layers for fine-tuning might be required to push classification performance significantly past 45% on these smaller 32x32 pixel images.

0.7 Code Appendix

Below is the source code orchestration used to train these models locally using macOS metal performance shaders (`mps`).

0.7.1 _targets.R

```
library("targets")
library("torch")
library("torchvision")
library("luz")
library("magrittr")

tar_option_set(
  packages = c("torch", "torchvision", "luz", "magrittr", "purrr", "dplyr")
)

tar_source("R/functions.R")

list(
```

```
  # 1. Dataset generation
  tar_target(fashion_train_ds, get_fashion_mnist_data(train = TRUE)),
  tar_target(fashion_test_ds, get_fashion_mnist_data(train = FALSE)),

  tar_target(mnist_train_ds, get_mnist_data(train = TRUE)),
  tar_target(mnist_test_ds, get_mnist_data(train = FALSE)),

  tar_target(cifar_train_ds, get_cifar10_data(train = TRUE)),
  tar_target(cifar_test_ds, get_cifar10_data(train = FALSE)),

  # 2. Part 1: Fashion MNIST
  tar_target(fashion_mnist_model, train_cnn_model(fashion_net, fashion_train_ds, fashion_tes
  tar_target(fashion_mnist_eval, evaluate_model(fashion_mnist_model)),

  # 3. Part 2: LeNet MNIST
  tar_target(lenet_mnist_model, train_cnn_model(lenet_5, mnist_train_ds, mnist_test_ds, epoc
  tar_target(lenet_mnist_eval, evaluate_model(lenet_mnist_model)),

  # 4. Part 3: LeNet CIFAR-10
  tar_target(lenet_cifar_model, train_lenet_cifar(cifar_train_ds, cifar_test_ds)),
  tar_target(lenet_cifar_eval, evaluate_model(lenet_cifar_model)),

  # 5. Part 4: VGG CIFAR-10
  tar_target(vgg_cifar_model, train_cnn_model(vgg_module, cifar_train_ds, cifar_test_ds, epo
  tar_target(vgg_cifar_eval, evaluate_model(vgg_cifar_model)),

  # 6. Part 5: Transfer Learning
  tar_target(transfer_learning_model, train_cnn_model(resnet_tl_module, cifar_train_ds, cifa
  tar_target(transfer_learning_eval, evaluate_model(transfer_learning_model))
)
```

0.7.2 R/functions.R

```
suppressPackageStartupMessages({
  library("torch")
  library("torchvision")
  library("luz")
  library("magrittr")
  library("purrr")
  library("dplyr")
})
```

```
# 1. Device Helper
get_device <- function() {
  if (cuda_is_available()) {
    torch_device("cuda")
  } else if (backends_mps_is_available()) {
    torch_device("mps")
  } else {
    torch_device("cpu")
  }
}


# 2. Data Loaders
get_fashion_mnist_data <- function(train) {
  fashion_mnist_dataset(
    root = "./data-fashion",
    train = train,
    download = TRUE,
    transform = function(x) {
      x %>%
        transform_to_tensor() %>%
        (function(t) if (length(t$shape) == 2) t$unsqueeze(1) else t) %>%
        transform_normalize(mean = 0.5, std = 0.5)
    }
  )
}

get_mnist_data <- function(train) {
  mnist_dataset(
    root = "./data-mnist",
    train = train,
    download = TRUE,
    transform = function(x) {
      x %>%
        transform_to_tensor() %>%
        (function(t) if (length(t$shape) == 2) t$unsqueeze(1) else t) %>%
        transform_normalize(mean = 0.5, std = 0.5)
    }
  )
}

get_cifar10_data <- function(train) {
  cifar10_dataset(
```

```
    root = "./data-cifar",
    train = train,
    download = TRUE,
    transform = function(x) {
      x %>%
        transform_to_tensor() %>%
        transform_normalize(mean = c(0.5, 0.5, 0.5), std = c(0.5, 0.5, 0.5))
    }
  )
}

# 3. Model Architectures
fashion_net <- nn_module(
  "FashionNet",
  initialize = function() {
    self$conv1 <- nn_conv2d(1, 32, kernel_size = 3)
    self$conv2 <- nn_conv2d(32, 64, kernel_size = 3)
    self$conv3 <- nn_conv2d(64, 64, kernel_size = 3)
    self$fc1 <- nn_linear(576, 64)
    self$fc2 <- nn_linear(64, 10)
  },
  forward = function(x) {
    x <- if (x$dim() == 3) x$unsqueeze(2) else x
    x %>%
      self$conv1() %>% nnf_relu() %>% nnf_max_pool2d(2) %>%
      self$conv2() %>% nnf_relu() %>% nnf_max_pool2d(2) %>%
      self$conv3() %>% nnf_relu() %>%
      torch_flatten(start_dim = 2) %>%
      self$fc1() %>% nnf_relu() %>%
      self$fc2() %>% nnf_softmax(dim = 2)
  }
)

lenet_5 <- nn_module(
  "LeNet5",
  initialize = function(channels = 1, output_nodes = 256) {
    self$conv1 <- nn_conv2d(channels, 6, kernel_size = 5)
    self$conv2 <- nn_conv2d(6, 16, kernel_size = 5)
    self$fc1 <- nn_linear(output_nodes, 120)
    self$fc2 <- nn_linear(120, 84)
    self$fc3 <- nn_linear(84, 10)
  },
  forward = function(x) {
```

```
    x <- if (x$dim() == 3) x$unsqueeze(2) else x
    x %>%
      self$conv1() %>% nnf_relu() %>% nnf_max_pool2d(2) %>%
      self$conv2() %>% nnf_relu() %>% nnf_max_pool2d(2) %>%
      torch_flatten(start_dim = 2) %>%
      self$fc1() %>% nnf_relu() %>%
      self$fc2() %>% nnf_relu() %>%
      self$fc3()
  }
)

vgg_module <- nn_module(
  "VGG",
  initialize = function() {
    self$features <- nn_sequential(
      nn_conv2d(3, 64, 3, padding = 1), nn_relu(inplace = TRUE),
      nn_conv2d(64, 64, 3, padding = 1), nn_relu(inplace = TRUE),
      nn_max_pool2d(2, 2),
      nn_conv2d(64, 128, 3, padding = 1), nn_relu(inplace = TRUE),
      nn_conv2d(128, 128, 3, padding = 1), nn_relu(inplace = TRUE),
      nn_max_pool2d(2, 2),
      nn_conv2d(128, 256, 3, padding = 1), nn_relu(inplace = TRUE),
      nn_conv2d(256, 256, 3, padding = 1), nn_relu(inplace = TRUE),
      nn_conv2d(256, 256, 3, padding = 1), nn_relu(inplace = TRUE),
      nn_max_pool2d(2, 2),
      nn_conv2d(256, 512, 3, padding = 1), nn_relu(inplace = TRUE),
      nn_conv2d(512, 512, 3, padding = 1), nn_relu(inplace = TRUE),
      nn_conv2d(512, 512, 3, padding = 1), nn_relu(inplace = TRUE),
      nn_max_pool2d(2, 2),
      nn_conv2d(512, 512, 3, padding = 1), nn_relu(inplace = TRUE),
      nn_conv2d(512, 512, 3, padding = 1), nn_relu(inplace = TRUE),
      nn_conv2d(512, 512, 3, padding = 1), nn_relu(inplace = TRUE),
      nn_max_pool2d(2, 2)
    )
    self$classifier <- nn_sequential(
      nn_linear(512, 4096), nn_relu(TRUE), nn_dropout(),
      nn_linear(4096, 4096), nn_relu(TRUE), nn_dropout(),
      nn_linear(4096, 10)
    )
  },
  forward = function(x) {
    x %>%
      self$features() %>%
```

```
      torch_flatten(start_dim = 2) %>%
      self$classifier()
  }
)

resnet_tl_module <- nn_module(
  "ResNet18_TL",
  initialize = function() {
      self$resnet <- model_resnet18(pretrained = TRUE)
      self$resnet$parameters %>% purrr::walk(function(param) param$requires_grad_(FALSE))
      self$resnet$fc <- nn_linear(512, 10)
  },
  forward = function(x) {
      self$resnet(x)
  }
)


# 4. Training Function Helper
train_cnn_model <- function(model_factory, train_ds, test_ds, epochs=5, lr=0.001, optimizer=o
  train_dl <- dataloader(train_ds, batch_size = batch_size, shuffle = TRUE)
  test_dl <- dataloader(test_ds, batch_size = batch_size)

  device <- get_device()
  message(sprintf("Training on %s", device$type))

  opt_list <- list(lr = lr, ...)

  fitted <- model_factory %>%
    setup(
      optimizer = optimizer,
      loss = nn_cross_entropy_loss(),
      metrics = list(luz_metric_accuracy())
    )

  fitted <- do.call(set_opt_hparams, c(list(fitted), opt_list))

  fitted <- fitted %>%
    fit(train_dl, epochs = epochs, valid_data = test_dl)


  state <- as.list(fitted$model$cpu()$state_dict())
  state_r <- lapply(state, function(x) as.array(x))
```

```
  list(
    metrics = get_metrics(fitted),
    state = state_r
  )
}

train_lenet_cifar <- function(train_ds, test_ds) {
  train_dl <- dataloader(train_ds, batch_size = 64, shuffle = TRUE)
  test_dl <- dataloader(test_ds, batch_size = 64)
  device <- get_device()

  fitted <- lenet_5 %>%
    setup(
      optimizer = optim_adam,
      loss = nn_cross_entropy_loss(),
      metrics = list(luz_metric_accuracy())
    ) %>%
    set_hparams(channels = 3, output_nodes = 400) %>%
    set_opt_hparams(lr = 0.001) %>%
    fit(train_dl, epochs = 120, valid_data = test_dl)

  state <- as.list(fitted$model$cpu()$state_dict())
  state_r <- lapply(state, function(x) as.array(x))

  list(metrics = get_metrics(fitted), state = state_r)
}

# 5. Evaluation Function Helper
evaluate_model <- function(trained_model) {
  metrics_df <- trained_model$metrics

  # Return the metrics data frame directly so we can inspect or plot it
  metrics_df
}
```