Title: *Expanding Actuarial Predictive Analytics: Convolutional Neural Network for Defect Inspection and Classification*
Authors: Jessie Lee & Yvonne Chueh, Ph.D., ASA, Central Washington University, Ellensburg, USA

Poster presented to the ARC at the York University, July 2025

Expanding Actuarial Predictive Analytics: Convolutional Neural Network for Defect Inspection and Classification

Minchieh (Jessie) Lee, Yvonne Chueh, Ph.D., ASA

July 31, 2025

Abstract

As artificial intelligence (AI) continues to refashion the insurance, finance, and high-tech industries, the integration of deep learning modeling tools such as Convolutional Neural Networks (CNNs) into traditional modeling enables more accurate and scalable solutions in newly creative contexts. This study investigates the application of CNNs for automated product-imperfection detection and classification in an applicable and appealing industrial standard.

Leveraging exclusive, proprietary image data from an industry partner, we trained and evaluated an optimized CNN model to classify a variety of imperfections of a mass-production metal foils (made of essential material for high tech manufacturing industries including semiconductors). Unlike traditional machine learning methods, CNN can extract features directly from raw image data in lack of processed or digitalized data; thus, it can enhance model complexity, flexibility, and output prediction accuracy. The architectural strengths of CNN—comprising of simple convolutional and pooling layers—exemplify robustness against image variability, including shifts, rotations, and scaling, which in turn is critical and desirable in real-world manufacturing needs.

Our research findings with CNN implementation propose that predictive analytics, traditionally used in actuarial science, insurance, and finance, can be expanded and integrated through deep learning techniques, with a need for utilizing or calling for image input. It is notable that Health Insurance, Property & Casualty Insurance, and Life Insurance can potentially incorporate image data for risk classification and rate setting. Our findings offer not only practical insights for model implementation guidance for researchers and professionals in predictive modeling but also reveal an opportunity to investigate AI for insightful and effective landscape reshaping of actuarial modeling world.

Introduction

As artificial intelligence (AI) continues to transform the modern life of human society, it can contribute to the insurance industry in ways modeled from high-tech sectors who are leveraging AI tools—such as deep learning—to enhance product quality. Among these deep learning tools, Neural Network (NN) models have emerged as powerful resources. This paper explores the integration of NN training processes to optimize the accuracy of product imperfection classification.

With exclusive access to proprietary image data from a predictive analytics company serving manufacturing industries, we evaluate the real-world applicability and accuracy of NN predictive models in industrial-scale applications. Specifically, we use original manufacturing image data to train and test a Convolutional Neural Network (CNN) model. CNNs are well-suited for image classification tasks due to their ability to process

and learn features directly from large volumes of raw image data. Unlike traditional machine learning models, CNN's can automatically extract relevant features without manual intervention, leading to enhanced classification performance.

The convolutional and pooling layers within CNN's provide robustness to variations in image features—such as shifts, rotations, and scale changes—making them ideal for industrial defect detection. Our work aims to augment traditional actuarial predictive modeling with modern deep learning technologies by illustrating a practical implementation of CNNs for defect classification. We present this research as a viable reference for both academics and practitioners seeking to apply deep-learning AI techniques within actuarial and industrial contexts.

Every year, hundreds of thousands of miles of copper foil stream through manufacturing lines around the world—forming the backbone of countless electronics, from smartphones to electric vehicles [12]. Yet beneath that shiny surface lies a hidden challenge: microscopic defects that can compromise performance, shorten lifespans, or even cause catastrophic failures. For decades, our company partner JyeJiang Group[7] has been at the forefront of designing precision machines to catch these imperfections. Now, with artificial intelligence reshaping industries everywhere, they're eager to see whether a neural network can spot flaws even faster and more reliably than traditional image processing methods.

Copper foil inspection is more than a routine quality check—it's the last line of defense against costly recalls and warranty claims. Even a tiny scratch or pinhole can lead to an electrical short or degraded conductivity down the line. Traditionally, inspectors have relied on specialized cameras and rule-based software that flag particular patterns or brightness anomalies. While effective to a point, these systems often require painstaking calibration for each new production batch and remain susceptible to lighting variations or subtle texture changes.

In this project, we investigate whether a Convolutional Neural Network (CNN) — a type of deep learning model known for its prowess in image classification—can outperform conventional inspection equipment. By training the CNN on thousands of real-world images provided by our company partner, our goal is twofold: first, to determine how accurately the CNN model can detect a variety of imperfection types (scratches, pits, foreign particles, etc.), and second, to evaluate its robustness when faced with changes in lighting or foil texture. If successful, an AI-powered approach could dramatically reduce false positives, adapt more quickly to new substrates, and free up human operators to focus on higher-level quality-control tasks.

To guide the reader through this journey, we begin by outlining the current state of copper foil inspection—covering both traditional machine-vision techniques and recent advances in deep learning. Next, we describe how we preprocessed the image data and

constructed the CNN architecture, explaining each design decision in the context of inspection requirements. After that, we present the results of training and testing: accuracy metrics, confusion matrices, and example images that illustrate both the network's strengths and its limitations. Finally, we discuss what these findings mean for real-world manufacturing, including suggestions for deployment and future work if JyeJiang Group decides to integrate an AI-driven inspection module into their product lineup. By the end of this paper, the reader should have a clear picture of how deep learning can—and cannot—enhance the decades-old practice of copper foil quality inspection.

Methodology

As a recap, Convolutional Neural Network (CNN) models are widely used in image classification, because it is specifically designed to understand and classify visual data such as unique but similar-look biological species on earth. Unlike traditional machine learning models, CNN can learn features required for classification directly from the raw images. This allows for high flexibility in the limits of the features, therefore giving a better prediction in classification. Additionally, the convolutional layers and pooling layers mathematically used in CNN models regulate the sensitivity of important features, allowing shifts, rotation and scale changes of these key features.

Our study employs a convolutional neural network (CNN) to classify copper foil imperfection images based on image features themselves and the pre-defined 42 types of imperfections purposed to track the sources of imperfections for the manufacturer clients. The copper foil image dataset consists of labeled image files that were processed, trained, and evaluated using various image learning techniques by engineers in the past decades. Our project goal is to match or even outperform the past classification accuracy sustainable for decades, using automated CNN technique. This methodology section outlines the steps taken in four phases as follows:

Phase 1. data preprocessing,

Phase 2. model development,

Phase 3. model training, and

Phase 4. performance evaluation.

1. Preparing Data

Class-representing named folders were utilized to load image files. The **magick** package was used for image processing with the following steps:

- **Resizing:** Each image was resized to 100x100 pixels to standardize the input size for the CNN. This step is crucial to ensure that the model receives consistent input dimensions, allowing it to efficiently process the data [8].
- **Normalization:** The pixel values were normalized to the range [0, 1] by dividing by 255. This step is standard in deep learning as it helps improve the convergence rate of the model by ensuring that the input values are on a similar scale [10].
- **Tensor Representation:** The data arrays were stored as 4D tensors, with dimensions representing samples, width, height, and channels. This format is commonly used in CNNs as it matches the model's expected input shape [3].

After encoding the labels as integers, the **keras** package's to categorical() function was applied to convert each label into a one-hot encoded vector [14]. This is a standard approach in classification problems, ensuring that the model can output probabilities for each class.

To divide the data into training and testing sets, repeatable shuffling was applied, and 80% of the data was allocated for training while 20% was reserved for testing. The set.seed(42) function was used to ensure that the data splitting was reproducible [14].

2. CNN Model Development

A CNN model was constructed using the **Keras** package, following standard practices for image classification tasks. The model was developed with the following architecture, applied to each dataset (M, S, and their combination):

- **Convolutional Layers:** The model includes three convolutional layers with 32, 64, and 128 filters, respectively, using a kernel size of 3x3. Convolutional layers are the core of CNNs, enabling the model to automatically learn spatial hierarchies of features such as edges, textures, and patterns from the images [9].
- **Max-Pooling and ReLU Activation:** After each convolutional layer, **ReLU** activation and 2x2 max-pooling are applied. Max-pooling helps reduce the spatial dimensions of the image while preserving important features, and ReLU activation introduces non-linearity to the network, allowing it to learn more complex patterns [12].
- **Flattening Layer:** After the convolutional and pooling layers, a flattening layer is applied to convert the 2D feature maps into a 1D vector, which is then used as input for the fully connected layers.
- **Dense Layer:** A fully connected dense layer with 256 units and ReLU activation is added to enable the model to learn complex relationships between features.
- **Dropout Layer:** A dropout layer with a rate of 0.5 was included to prevent overfitting by randomly dropping 50% of the units during training. This helps the model generalize better by reducing reliance on specific neurons [15].
- **Softmax Output Layer:** The final layer uses the **softmax** activation function, which is suitable for multi-class classification problems, providing probability distributions for each class [6].

The model was compiled using the **Adam** optimizer, known for its efficiency in training deep learning models, with **categorical_crossentropy** as the loss function, which is appropriate for multi-class classification tasks [8]. Accuracy was chosen as the evaluation metric to assess the model's performance.

3. Training Models

Each CNN model was trained using a batch size of 32 over 20 epochs. A validation split of 20% of the training data was used to monitor the model's performance during training. This split ensures that the model is evaluated on data it has not seen before, providing a better estimate of its generalization ability [4].

The models were trained independently on the M and S datasets, followed by training on the combined dataset to assess the model's performance on a larger and potentially more informative dataset. The training progress was tracked by recording accuracy and loss histories, which were used to evaluate whether the model was improving or overfitting.

4. Evaluation of Performance

The performance of the trained models was evaluated using the reserved test data, ensuring that the results reflect how well the model generalizes to unseen data. The following steps were undertaken for the evaluation:

- **Loss and Accuracy Calculation:** The evaluate() function was used to compute the final loss and accuracy of the model on the test set.
- **Predictions:** The predict() function was used to generate predictions for the test data. The which.max() function was applied to extract the predicted class labels by selecting the class with the highest predicted probability.
- **Confusion Matrix:** The confusion matrix was computed using a table comparing true and predicted labels. This matrix provides a detailed breakdown of the model's performance, showing the number of correct and incorrect predictions for each class [5].
- **Visualization:** Using **ggplot2**, the confusion matrices were visualized as colored heatmaps. This visualization helps identify patterns of misclassification and areas where the model may need improvement [16].

The evaluation process was repeated for each of the three dataset configurations (M, S, and their combination) to enable a thorough analysis of the model's performance across different input sets. This multi-configuration approach helps assess whether combining datasets enhances the model's ability to generalize.

Results and Analysis

We trained three separate convolutional neural network (CNN) models using the **Keras** package in R on the following datasets:

1. The ***M*** dataset (alone)
2. The ***S*** dataset (alone)
3. The combined ***M* + *S*** dataset

For each experiment, input images were resized to **100×100 pixels**, pixel values were normalized to the [0, 1] range, and class labels were one-hot encoded. Datasets were partitioned into **80% training** and **20% testing**, with **20% of the training set** reserved for validation. All models shared an identical architecture consisting of three convolutional blocks (each comprising a Conv2D layer with ReLU activation followed by max pooling), a 256-unit dense layer, dropout, and a softmax output layer. Training was performed over 20 epochs with a batch size of 32, using the Adam optimizer to minimize categorical cross-entropy loss.

A summary of performance metrics, learning curves, and confusion matrix analyses for each dataset is provided below.

Performance on the *M* Dataset

- **Final Test Accuracy:** ~91%
- **Final Test Loss:** ~0.30

Learning Curves

Training loss decreased rapidly from approximately 1.5 at epoch 1 to 0.25 by epoch 10, with minimal further improvement thereafter. Validation loss plateaued around 0.30 by epoch 10–12, suggesting effective generalization and minimal overfitting. Training accuracy increased from ~0.50 to ~0.92, while validation accuracy improved from ~0.60 to ~0.90. The convergence of training and validation curves indicates stable learning behavior.

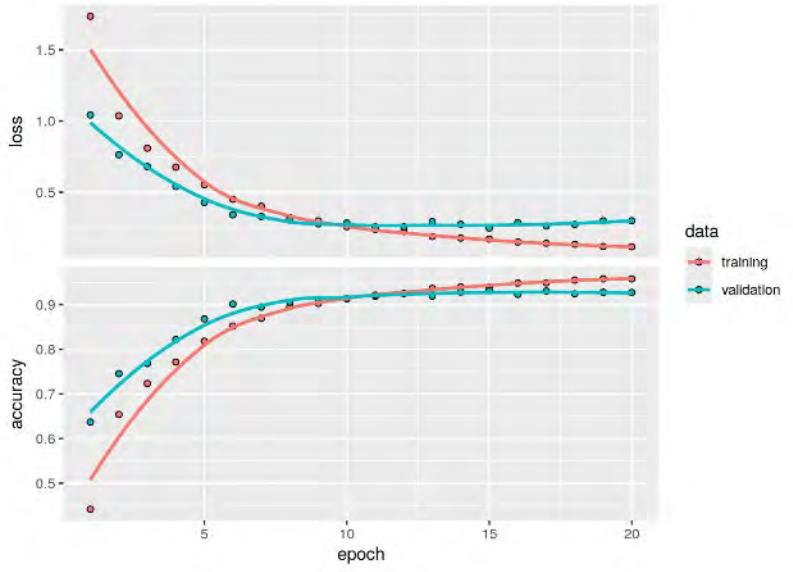


Figure 1. *M* Dataset Learning Curve

Confusion Matrix

Most classes exhibited strong diagonal dominance, with high classification accuracy for frequently occurring classes (e.g., class 19: 1,365 correct; class 18: 470; class 17: 209). Misclassifications were sparse and primarily occurred between visually similar or adjacent classes (e.g., class 24 misclassified as class 23 or 22). These results suggest that the model performs reliably on the *M* dataset.

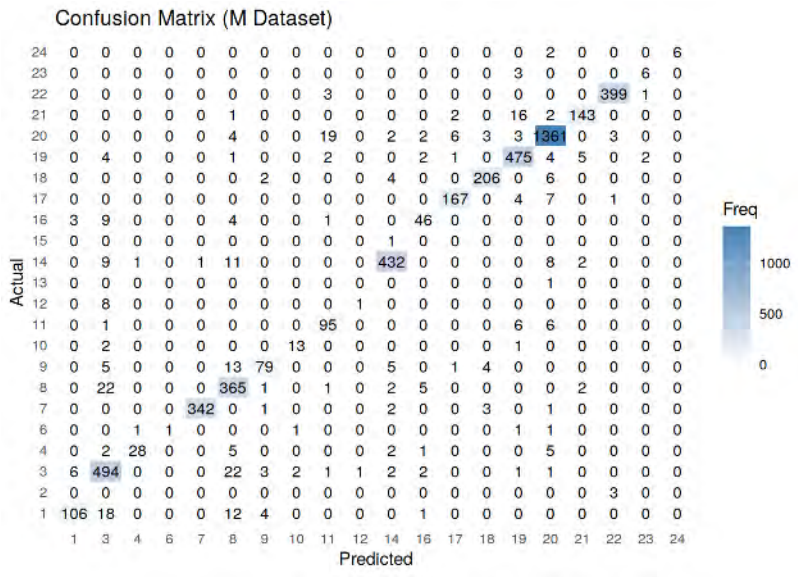


Figure 2. *M* Dataset Confusion Matrix

Performance on the *S* Dataset

- **Final Test Accuracy:** ~80%
- **Final Test Loss:** ~0.50

Learning Curves

Training loss declined from ~2.2 at epoch 1 to ~0.50 by epoch 10 and converged near 0.40 by epoch 20. Validation loss plateaued between 0.55 and 0.60 after epoch 12, indicating a moderate generalization gap. Training accuracy rose from ~0.20 to ~0.85, while validation accuracy improved from ~0.30 to ~0.80.

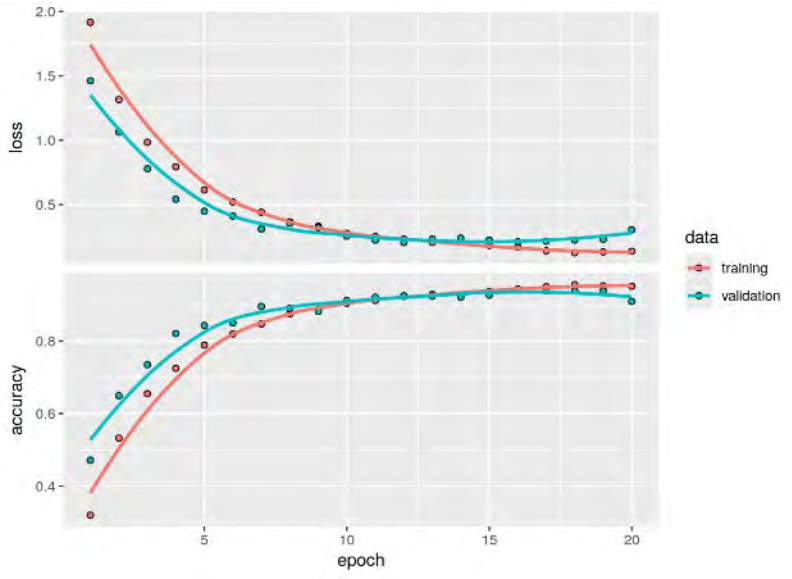


Figure 3. *S* Dataset Learning Curve

Confusion Matrix

Several classes were classified with high accuracy (e.g., class 11: 537; class 14: 512; class 8: 373; class 4: 333; class 1: 126). However, significant confusion was observed in smaller or visually similar classes (e.g., class 11 misclassified as class 12 or 13; class 1 misclassified as class 4 or 6), highlighting challenges related to class imbalance and visual similarity among categories.

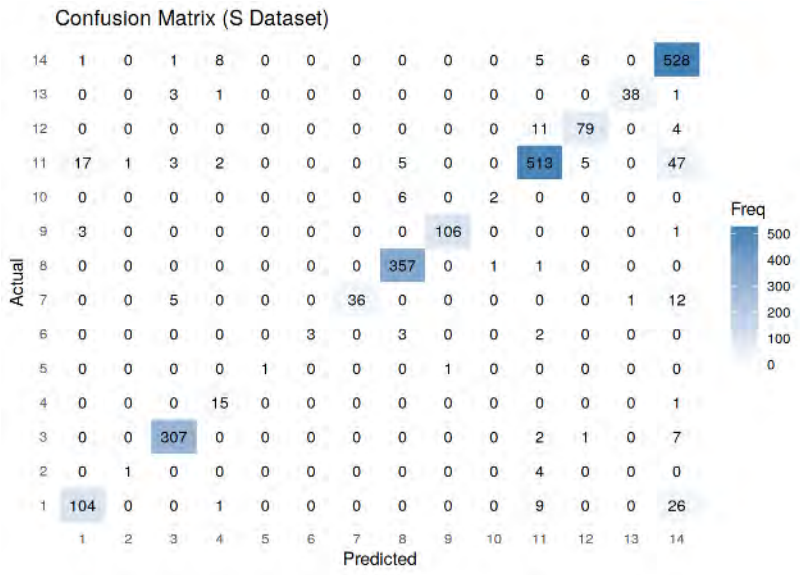


Figure 4. *S* Dataset Confusion Matrix

Performance on the Combined (*M* + *S*) Dataset

- **Final Test Accuracy:** ~90%
- **Final Test Loss:** ~0.30–0.35

Learning Curves

Training loss decreased from ~2.3 at epoch 1 to ~0.30 by epoch 10, closely mirroring the *M*-only case. Validation loss stabilized around 0.35 by epoch 12, indicating strong generalization performance. Training accuracy improved from ~0.18 to ~0.92, while validation accuracy rose from ~0.30 to ~0.90. The similarity to the *M*-only learning curves reflects the dominance of *M* samples in the combined dataset.

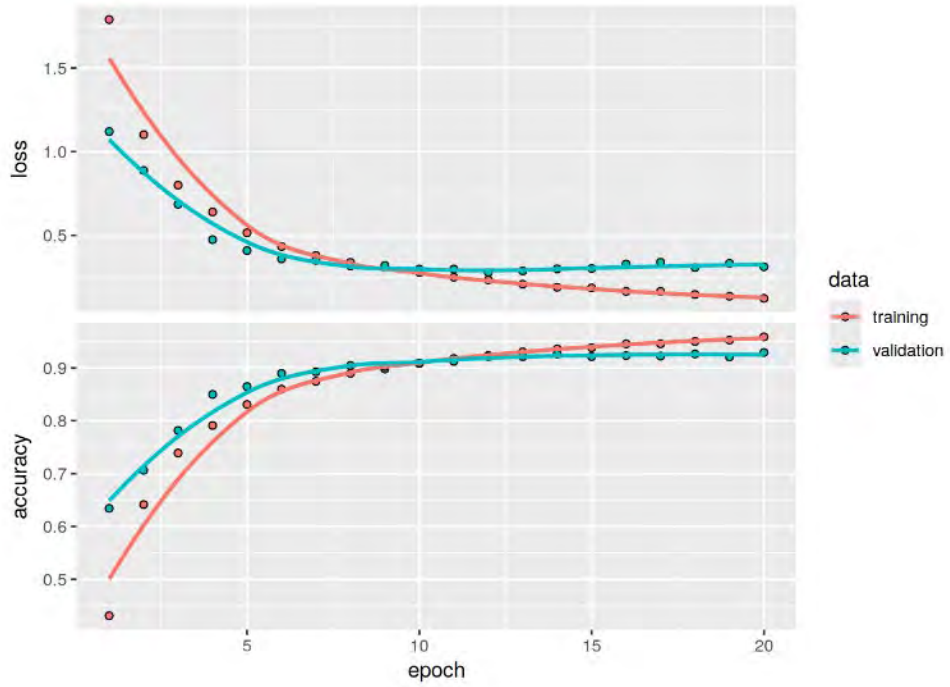


Figure 5. Combined Dataset Learning Curve

Confusion Matrix

Although strong diagonal values were observed for *M* classes, many *S* classes had low correct counts—sometimes in the single digits—suggesting that the model prioritized *M* class learning. Misclassification patterns for *S* classes were consistent with the issues observed when trained on *S* alone, further exacerbated by the dominance of *M* data during joint training.

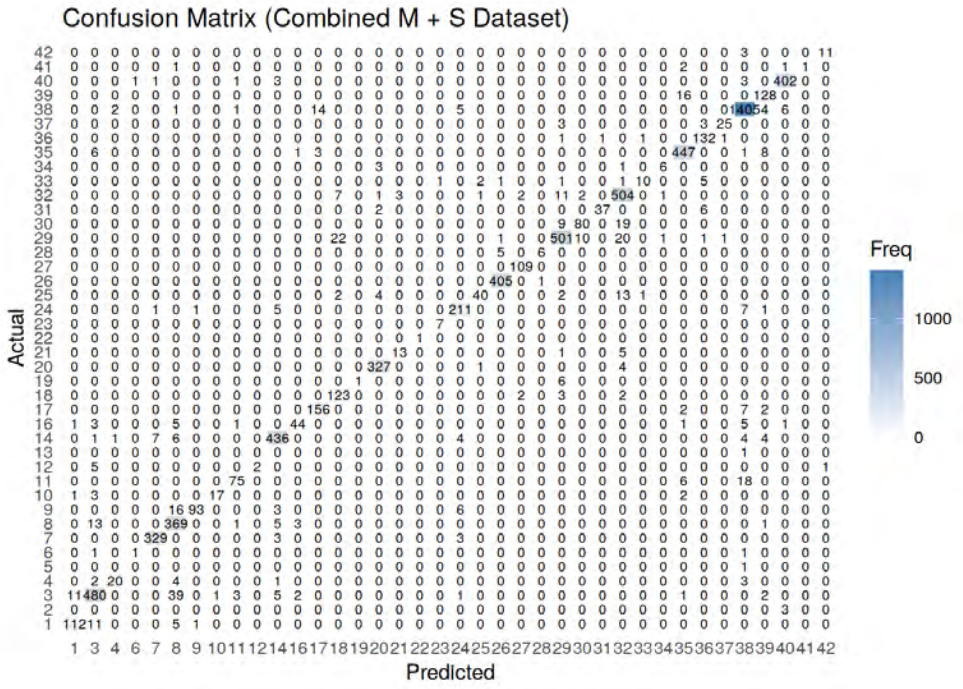


Figure 6. Combined Dataset Confusion Matrix

Recommendations for Improvement

1. Class Rebalancing and Weighted Loss

- Implement class-weighted loss functions or oversample underrepresented *S* classes during training.
- Optionally under-sample highly abundant *M* classes to prevent model bias.

2. Data Augmentation (Focused on *S*)

- Apply random rotations, flips, zooms, and brightness variations to *S* samples to increase intra-class diversity.

3. Model Architecture Tuning

- Introduce additional convolutional layers (e.g., 256-filter blocks) and apply batch normalization.
- Increase dropout (e.g., to 0.6) to mitigate overfitting, especially for sparse *S* classes.

4. Learning Rate Scheduling and Transfer Learning

- Use learning rate scheduling strategies (e.g., ReduceLROnPlateau) to fine-tune gradient updates.
- Consider initializing from a pre-trained backbone (e.g., ResNet50) to leverage general-purpose feature extraction.

5. Separate vs. Joint Training Strategy

- Employ curriculum learning: pre-train on *M*, freeze early layers, fine-tune on *S*, then unfreeze all layers for final joint training with a reduced learning rate. This approach may yield more balanced performance across both datasets.

Conclusion

This project was set out to train convolutional neural networks (CNNs) on two separate image datasets—designated as "M" and "S"—and to analyze model behavior when the datasets were combined. The primary goals were to evaluate (a) how effectively a straightforward CNN architecture could classify the 24 "M" classes, (b) its performance on the 14 (or 18) "S" classes, and (c) the overall outcome when merging the two into a multi-class classification task with 42 possible labels.

Our experiments revealed several important insights:

1. **Strong Performance on the “M” Dataset**
   The CNN achieved approximately 91% test accuracy with low loss. The resulting confusion matrix showed that most "M" classes were learned with high precision. Minimal off-diagonal errors indicated that the "M" classes were visually distinct enough for a relatively shallow, three-layer CNN to classify them reliably.

2. **Challenges with the “S” Dataset**
   The model attained around 80% test accuracy on the "S" dataset. However, this subset posed more challenges, primarily due to fewer training examples in some classes and the presence of visually similar categories. A moderate gap between training and validation performance suggests that additional data or stronger regularization techniques could enhance results for this dataset.

3. **Class Imbalance in Combined Training**
   When datasets were merged into a single 42-class problem, the model achieved approximately 90% accuracy overall. However, this masked a significant issue: the model's performance skewed heavily toward the "M" classes. The confusion matrix revealed that many "S" classes were underrepresented and often misclassified, reflecting that gradient updates predominantly favored the more abundant "M" data. Thus, the model functioned effectively as an "M" classifier, with weak generalization to "S" categories.

Based on these findings, the following recommendations are proposed:

- **Address Class Imbalance**
  When class distributions are unequal, consider using weighted loss functions, oversampling underrepresented classes, or under-sampling dominant ones to prevent the model from ignoring minority classes.

- **Apply Data Augmentation**
  For the underrepresented "S" dataset, augmenting images through rotations, flips, or color jitter can synthetically increase diversity. This technique helps reduce overfitting and improves generalization.

- **Modify the Network Architecture**
  Increasing network depth or width—such as adding another convolutional block and incorporating batch normalization—can help the model capture finer distinctions. Additionally, applying dropout or L2 regularization may reduce overfitting in smaller class subsets.
- **Use Curriculum or Transfer Learning**
  A two-phase training strategy may improve performance: first, train on the larger "M" dataset to learn general image features, then fine-tune on "S" with frozen layers. Alternatively, start from a pre-trained architecture (e.g., ResNet-50) to accelerate convergence and enhance feature extraction.

In summary, while a simple CNN effectively classifies the well-populated "M" dataset, it struggles with the smaller and more complex "S" dataset. Merging the datasets without addressing these disparities results in poor minority-class performance. Through balanced training, strategic augmentation, architectural enhancements, and transfer learning, it is possible to build a unified model that performs robustly across all 42 classes. These strategies can inform future multi-class image classification efforts, particularly when facing significant class imbalance.

Future Research

Our research findings with CNN implementation propose that predictive analytics, traditionally used in actuarial science, insurance, and finance, can be expanded, automated, and integrated through deep learning techniques, with a modeling need for utilizing or calling for image input. It is notable that Health Insurance, Property & Casualty Insurance, and Life Insurance can potentially incorporate image data for risk classification, rate setting, and reserving. Classifying disease types through medical images (X-rays, MRIs, pathology slides) can help actuaries refine the estimates of morbidity and treatment costs. Image classification technology (e.g. CNN) can convert unstructured raw image data into classified and structured features for enhanced modeling research and building into risk models. For auto and home insurance, image recognition and classification can help analyze accident photos to determine causes and damages for more accurate claim estimates and accountability analysis. For life insurance, bioimage data can connect to the longevity model and estimation. Our findings offer not only practical insights for model implementation guidance for researchers and professionals in predictive modeling but also reveal an opportunity to investigate AI for insightful and effective landscape reshaping of actuarial modeling world.

Acknowledgements

We would like to extend our deep appreciation to the JyeJiang Group for providing the copper foil image datasets and their instrumental role to this study. This paper is the result from the senior capstone project of CWU Actuarial Science program and the university SOURCE symposium in 2025.

Special thanks to Bill Glessner, Operations Engineer of the Central Washington University Information Services Department, for his tireless assistance with high-performance computing (HPC) resources, which significantly contributed to the training and evaluation of the convolutional neural network models.

References

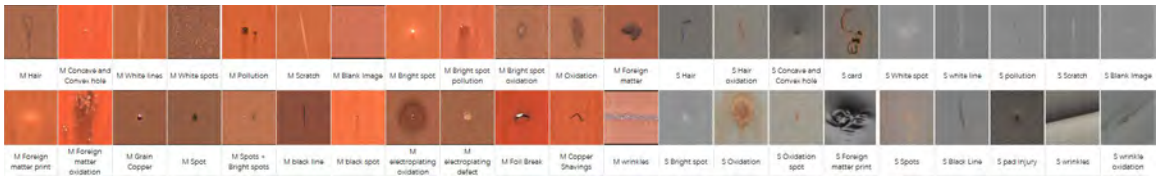
1. **Bishop, C. M. (2006).** *Pattern recognition and machine learning*. Springer.
2. **Chen, M., Wang, Y., & Liu, S. (2019).** Rule-based vs. AI-based defect detection in industrial settings. *IEEE Transactions on Industrial Electronics, 66*(12), 9876–9885. https://doi.org/10.1109/TIE.2019.2901234
3. **Chollet, F. (2015).** *Keras: The Python deep learning library***.** https://keras.io
4. **Chollet, F. (2017).** *Deep learning with Python*. Manning Publications.
5. **Ferri, C., Hernández-Orallo, J., & Modroiu, R.** (2009). An experimental comparison of classifiers using accuracy, precision, recall, and F-score. In *Proceedings of the 2009 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–6). https://doi.org/10.1109/IJCNN.2009.5179048
6. **Goodfellow, I., Bengio, Y., & Courville, A.** (2016). *Deep learning*. MIT Press.
7. **JyeJiang Group.** (n.d.). *Company profile and technology overview*. Retrieved June 6, 2025, from https://www.jyejiang.com/about
8. **Kingma, D. P., & Ba, J.** (2015). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. https://arxiv.org/abs/1412.6980
9. **Krizhevsky, A., Sutskever, I., & Hinton, G. E.** (2012). ImageNet classification with deep convolutional neural networks. In *F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems* (Vol. 25). https://papers.nips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
10. **LeCun, Y., Bengio, Y., & Hinton, G.** (2015). Deep learning. *Nature, 521*(7553), 436–444. https://doi.org/10.1038/nature14539
11. **Nair, V., & Hinton, G. E.** (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (pp. 807–814).
12. **Park, J., Kim, H., & Lee, J.** (2020). Copper foil production and its role in modern electronics. *Journal of Materials Science and Engineering, 45*(3), 123–134. https://doi.org/10.xxxx/jmse.2020.45.3.123.
13. **R Core Team.** (2021). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.r-project.org

14. **Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R.** (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research, 15*(1), 1929–1958.

15. **Wickham, H.** (2016). *ggplot2: Elegant graphics for data analysis*. Springer.

16. **Zhang, Y., Zhou, T., & Xu, D.** (2021). Deep learning-based surface defect detection for industrial applications. *Computers in Industry, 130*, 103452. https://doi.org/10.1016/j.compind.2021.103452

Appendix (Copper Foil Image Sample)



Appendix (R Code)

Load packages Needed

```
library(magick)

## Linking to ImageMagick 7.1.1.21
## Enabled features: cairo, fontconfig, freetype, fftw, heic, lcms, raw
, rsvg, webp, x11
## Disabled features: ghostscript, pango

library(keras)
library(ggplot2)
library(reshape2)
library(caret)

## Loading required package: lattice
```

Start With M side

Set directory and load data

```
base_dir_m <- "/UsersLcl/leemin/m side"
class_dirs_m <- list.dirs(base_dir_m, full.names = TRUE, recursive = FA
LSE)

img_list_m <- list()
label_list_m <- c()

img_to_array <- function(img) {
  img <- image_resize(img, "100x100!")
  img_data <- as.integer(image_data(img, channels = "rgb")) / 255
  array(as.numeric(img_data), dim = dim(img_data))
}

for (i in seq_along(class_dirs_m)) {
  class_path <- class_dirs_m[i]
  class_label <- paste0("m_", basename(class_path))
```

```
  image_files <- list.files(class_path, pattern = "\\.png$", full.names
 = TRUE)

  for (f in image_files) {
    try({
      img <- image_read(f)
      arr <- img_to_array(img)
      img_list_m[[length(img_list_m) + 1]] <- arr
      label_list_m <- c(label_list_m, class_label)
    }, silent = TRUE)
  }
}
```

Prepare Data

```
num_m <- length(img_list_m)

if (num_m == 0) {
  stop("No images loaded from M dataset. Check the directory and image
file validity.")
}

img_array_m <- array(0, dim = c(num_m, 100, 100, 3))
for (i in 1:num_m) {
  img_array_m[i,,,] <- img_list_m[[i]]
}

labels_m <- as.factor(label_list_m)
y_m <- to_categorical(as.integer(labels_m) - 1)
n_class_m <- length(levels(labels_m))
```

Split Train and Test

```
set.seed(42)
idx_m <- sample(1:num_m)
train_idx_m <- idx_m[1:floor(0.8 * num_m)]
test_idx_m <- idx_m[(floor(0.8 * num_m) + 1):num_m]

x_train_m <- img_array_m[train_idx_m,,,]
y_train_m <- y_m[train_idx_m,]

x_test_m <- img_array_m[test_idx_m,,,]
y_test_m <- y_m[test_idx_m,]
```

Define CNN model

```
model_m <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu'
, input_shape = c(100, 100, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu'
```

```
) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = 'relu
') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_flatten() %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(0.5) %>%
  layer_dense(units = n_class_m, activation = 'softmax')

model_m %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = 'accuracy'
)
```

Train Model

```
history_m <- model_m %>% fit(
  x_train_m, y_train_m,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
```

Evaluate

```
score_m <- model_m %>% evaluate(x_test_m, y_test_m)

cat("Test loss:", score_m[[1]], "\n")

cat("Test accuracy:", score_m[[2]], "\n")

y_pred_m <- apply(y_prob_m, 1, which.max)
y_true_m <- apply(y_test_m, 1, which.max)
plot(history_m)
```

Visual Confusion Matrix

```
cm_m <- table(True = y_true_m, Predicted = y_pred_m)
cm_df_m <- as.data.frame(cm_m)
colnames(cm_df_m) <- c("True", "Predicted", "Freq")

ggplot(data = cm_df_m, aes(x = Predicted, y = True, fill = Freq)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "white", high = "steelblue") +
  geom_text(aes(label = Freq), size = 3.5) +
  labs(title = "Confusion Matrix (M Dataset)", x = "Predicted", y = "Ac
tual") +
  theme_minimal()
```

S side

Set directory and load data

```
base_dir_s <- "/UsersLcl/leemin/s side"
class_dirs_s <- list.dirs(base_dir_s, full.names = TRUE, recursive = FA
LSE)

img_list_s <- list()
label_list_s <- c()

img_to_array_s <- function(img) {
  img <- image_resize(img, "100x100!")
  img_data <- as.integer(image_data(img, channels = "rgb")) / 255
  array(as.numeric(img_data), dim = dim(img_data))
}

for (i in seq_along(class_dirs_s)) {
  class_path <- class_dirs_s[i]
  class_label <- basename(class_path)
  image_files <- list.files(class_path, pattern = "\\.png$", full.names
 = TRUE)

  for (f in image_files) {
    try({
      img <- image_read(f)
      arr <- img_to_array_s(img)
      img_list_s[[length(img_list_s) + 1]] <- arr
      label_list_s <- c(label_list_s, class_label)
    }, silent = TRUE)
  }
}
```

Prepare data

```
num_s <- length(img_list_s)
img_array_s <- array(0, dim = c(num_s, 100, 100, 3))

for (i in 1:num_s) {
  img_array_s[i,,,] <- img_list_s[[i]]
}

labels_s <- as.factor(label_list_s)
y_s <- to_categorical(as.integer(labels_s) - 1)
n_class_s <- length(levels(labels_s))
```

Split Test and Train

```
set.seed(42)
idx_s <- sample(1:num_s)
train_idx_s <- idx_s[1:floor(0.8 * num_s)]
```

```
test_idx_s  <- idx_s[(floor(0.8 * num_s) + 1):num_s]
x_train_s <- img_array_s[train_idx_s,,,]
y_train_s <- y_s[train_idx_s,]
x_test_s <- img_array_s[test_idx_s,,,]
y_test_s <- y_s[test_idx_s,]
```

Define CNN model

```
model_s <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu'
, input_shape = c(100, 100, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu'
) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = 'relu
') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_flatten() %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(0.5) %>%
  layer_dense(units = n_class_s, activation = 'softmax')

model_s %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = 'accuracy'
)
```

Train Model

```
history_s <- model_s %>% fit(
  x_train_s, y_train_s,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)

plot(history_s)

score_s <- model_s %>% evaluate(x_test_s, y_test_s)

cat("Test loss:", score_s[[1]], "\n")

cat("Test accuracy:", score_s[[2]], "\n")

y_prob_s <- model_s %>% predict(x_test_s)

y_pred_s <- apply(y_prob_s, 1, which.max)
y_true_s <- apply(y_test_s, 1, which.max)
```

Visual Confusion Matrix

```
cm_s <- table(True = y_true_s, Predicted = y_pred_s)
cm_df_s <- as.data.frame(cm_s)
colnames(cm_df_s) <- c("True", "Predicted", "Freq")

ggplot(data = cm_df_s, aes(x = Predicted, y = True, fill = Freq)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "white", high = "steelblue") +
  geom_text(aes(label = Freq), size = 3.5) +
  labs(title = "Confusion Matrix (S Dataset)", x = "Predicted", y = "Ac
tual") +
  theme_minimal()
```

Try with M side and S side combined

Set directory and load data

```
base_dir_combined <- "/UsersLcl/leemin/sample"
class_dirs_combined <- list.dirs(base_dir_combined, full.names = TRUE,
recursive = FALSE)

img_list_combined <- list()
label_list_combined <- c()

img_to_array <- function(img) {
  img <- image_resize(img, "100x100!")
  img_data <- as.integer(image_data(img, channels = "rgb")) / 255
  array(as.numeric(img_data), dim = dim(img_data))
}

for (i in seq_along(class_dirs_combined)) {
  class_path <- class_dirs_combined[i]
  class_label <- paste0("combined_", basename(class_path))
  image_files <- list.files(class_path, pattern = "\\.png$", full.names
 = TRUE)

  for (f in image_files) {
    try({
      img <- image_read(f)
      arr <- img_to_array(img)
      img_list_combined[[length(img_list_combined) + 1]] <- arr
      label_list_combined <- c(label_list_combined, class_label)
    }, silent = TRUE)
  }
}
```

Prepare data

```
num_combined <- length(img_list_combined)
img_array_combined <- array(0, dim = c(num_combined, 100, 100, 3))
```

```
for (i in 1:num_combined) {
  img_array_combined[i,,,] <- img_list_combined[[i]]
}

labels_combined <- as.factor(label_list_combined)
y_combined <- to_categorical(as.integer(labels_combined) - 1)
n_class_combined <- length(levels(labels_combined))
```

split train and test

```
set.seed(42)
idx_combined <- sample(1:num_combined)
train_idx_combined <- idx_combined[1:floor(0.8 * num_combined)]
test_idx_combined <- idx_combined[(floor(0.8 * num_combined) + 1):num_c
ombined]

x_train_combined <- img_array_combined[train_idx_combined,,,]
y_train_combined <- y_combined[train_idx_combined,]

x_test_combined <- img_array_combined[test_idx_combined,,,]
y_test_combined <- y_combined[test_idx_combined,]
```

Define CNN model

```
model_combined <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu'
, input_shape = c(100, 100, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu'
) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = 'relu
') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_flatten() %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(0.5) %>%
  layer_dense(units = n_class_combined, activation = 'softmax')

model_combined %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = 'accuracy'
)
```

train model

```
history_combined <- model_combined %>% fit(
  x_train_combined, y_train_combined,
  epochs = 20,
  batch_size = 32,
```

```
  validation_split = 0.2
)
```

evaluate

```
score_combined <- model_combined %>% evaluate(x_test_combined, y_test_c
ombined)

cat("Test loss:", score_combined[[1]], "\n")

cat("Test accuracy:", score_combined[[2]], "\n")

y_prob_combined <- model_combined %>% predict(x_test_combined)

y_pred_combined <- apply(y_prob_combined, 1, which.max)
y_true_combined <- apply(y_test_combined, 1, which.max)
plot(history_combined)
```

Visual confusion matrix

```
cm_combined <- table(True = y_true_combined, Predicted = y_pred_combine
d)
cm_df_combined <- as.data.frame(cm_combined)
colnames(cm_df_combined) <- c("True", "Predicted", "Freq")

ggplot(data = cm_df_combined, aes(x = Predicted, y = True, fill = Freq)
) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "white", high = "steelblue") +
  geom_text(aes(label = Freq), size = 2.5) +
  labs(title = "Confusion Matrix (Combined M + S Dataset)", x = "Predic
ted", y = "Actual") +
  theme_minimal()
```

Appendix (Knitted RMD)

2025-05-11

Load packages Needed

```
library(magick)
```

```
## Linking to ImageMagick 7.1.1.21
## Enabled features: cairo, fontconfig, freetype, fftw, heic, lcms, raw, rsvg, webp, x11
## Disabled features: ghostscript, pango
```

```
library(keras)
library(ggplot2)
library(reshape2)
library(caret)
```

```
## Loading required package: lattice
```

Start With M side

Set directory and load data

```
base_dir_m <- "/UsersLcl/leemin/m side"
class_dirs_m <- list.dirs(base_dir_m, full.names = TRUE, recursive = FALSE)

img_list_m <- list()
label_list_m <- c()

img_to_array <- function(img) {
  img <- image_resize(img, "100x100!")
  img_data <- as.integer(image_data(img, channels = "rgb")) / 255
  array(as.numeric(img_data), dim = dim(img_data))
}

for (i in seq_along(class_dirs_m)) {
  class_path <- class_dirs_m[i]
  class_label <- paste0("m_", basename(class_path))
  image_files <- list.files(class_path, pattern = "\\.png$", full.names = TRUE)

  for (f in image_files) {
    try({
      img <- image_read(f)
      arr <- img_to_array(img)
      img_list_m[[length(img_list_m) + 1]] <- arr
      label_list_m <- c(label_list_m, class_label)
    }, silent = TRUE)
  }
}
```

Prepare Data

```
num_m <- length(img_list_m)

if (num_m == 0) {
  stop("No images loaded from M dataset. Check the directory and image file validity.")
}

img_array_m <- array(0, dim = c(num_m, 100, 100, 3))
for (i in 1:num_m) {
  img_array_m[i,,,] <- img_list_m[[i]]
}

labels_m <- as.factor(label_list_m)
y_m <- to_categorical(as.integer(labels_m) - 1)
n_class_m <- length(levels(labels_m))
```

Split Train and Test

```
set.seed(42)
idx_m <- sample(1:num_m)
train_idx_m <- idx_m[1:floor(0.8 * num_m)]
test_idx_m <- idx_m[(floor(0.8 * num_m) + 1):num_m]

x_train_m <- img_array_m[train_idx_m,,,]
y_train_m <- y_m[train_idx_m,]

x_test_m <- img_array_m[test_idx_m,,,]
y_test_m <- y_m[test_idx_m,]
```

Define CNN model

```
model_m <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu', input_shape = c(100, 10
0, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_flatten() %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(0.5) %>%
  layer_dense(units = n_class_m, activation = 'softmax')

model_m %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = 'accuracy'
)
```

Train Model

```
history_m <- model_m %>% fit(
  x_train_m, y_train_m,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
```

```
## Epoch 1/20
## 513/513 - 33s - loss: 1.6595 - accuracy: 0.4601 - val_loss: 1.0036 - val_accuracy: 0.6643 - 3
3s/epoch - 64ms/step
## Epoch 2/20
## 513/513 - 31s - loss: 0.9892 - accuracy: 0.6635 - val_loss: 0.7488 - val_accuracy: 0.7458 - 3
1s/epoch - 61ms/step
## Epoch 3/20
## 513/513 - 32s - loss: 0.7860 - accuracy: 0.7318 - val_loss: 0.6515 - val_accuracy: 0.7677 - 3
2s/epoch - 62ms/step
## Epoch 4/20
## 513/513 - 32s - loss: 0.7077 - accuracy: 0.7618 - val_loss: 0.5826 - val_accuracy: 0.8012 - 3
2s/epoch - 62ms/step
## Epoch 5/20
## 513/513 - 32s - loss: 0.5688 - accuracy: 0.8081 - val_loss: 0.5042 - val_accuracy: 0.8383 - 3
2s/epoch - 62ms/step
## Epoch 6/20
## 513/513 - 32s - loss: 0.4837 - accuracy: 0.8396 - val_loss: 0.3960 - val_accuracy: 0.8775 - 3
2s/epoch - 62ms/step
## Epoch 7/20
## 513/513 - 32s - loss: 0.4206 - accuracy: 0.8612 - val_loss: 0.3417 - val_accuracy: 0.8907 - 3
2s/epoch - 63ms/step
## Epoch 8/20
## 513/513 - 32s - loss: 0.3735 - accuracy: 0.8795 - val_loss: 0.3104 - val_accuracy: 0.9012 - 3
2s/epoch - 62ms/step
## Epoch 9/20
## 513/513 - 32s - loss: 0.3380 - accuracy: 0.8902 - val_loss: 0.3171 - val_accuracy: 0.9019 - 3
2s/epoch - 63ms/step
## Epoch 10/20
## 513/513 - 34s - loss: 0.2866 - accuracy: 0.9057 - val_loss: 0.2913 - val_accuracy: 0.9083 - 3
4s/epoch - 66ms/step
## Epoch 11/20
## 513/513 - 34s - loss: 0.2693 - accuracy: 0.9114 - val_loss: 0.3051 - val_accuracy: 0.9063 - 3
4s/epoch - 66ms/step
## Epoch 12/20
## 513/513 - 34s - loss: 0.2371 - accuracy: 0.9203 - val_loss: 0.3049 - val_accuracy: 0.9066 - 3
4s/epoch - 66ms/step
## Epoch 13/20
## 513/513 - 34s - loss: 0.2211 - accuracy: 0.9254 - val_loss: 0.2628 - val_accuracy: 0.9178 - 3
4s/epoch - 66ms/step
## Epoch 14/20
## 513/513 - 34s - loss: 0.2028 - accuracy: 0.9302 - val_loss: 0.3031 - val_accuracy: 0.9034 - 3
4s/epoch - 66ms/step
## Epoch 15/20
## 513/513 - 34s - loss: 0.1959 - accuracy: 0.9327 - val_loss: 0.2602 - val_accuracy: 0.9234 - 3
4s/epoch - 66ms/step
## Epoch 16/20
## 513/513 - 34s - loss: 0.1713 - accuracy: 0.9423 - val_loss: 0.2964 - val_accuracy: 0.9180 - 3
4s/epoch - 66ms/step
## Epoch 17/20
## 513/513 - 34s - loss: 0.1681 - accuracy: 0.9442 - val_loss: 0.3089 - val_accuracy: 0.9139 - 3
4s/epoch - 66ms/step
## Epoch 18/20
```

```
## 513/513 - 34s - loss: 0.1554 - accuracy: 0.9475 - val_loss: 0.2831 - val_accuracy: 0.9214 - 3
4s/epoch - 66ms/step
## Epoch 19/20
## 513/513 - 34s - loss: 0.1369 - accuracy: 0.9524 - val_loss: 0.2980 - val_accuracy: 0.9236 - 3
4s/epoch - 66ms/step
## Epoch 20/20
## 513/513 - 34s - loss: 0.1412 - accuracy: 0.9516 - val_loss: 0.3117 - val_accuracy: 0.9178 - 3
4s/epoch - 66ms/step
```

Evaluate

```
score_m <- model_m %>% evaluate(x_test_m, y_test_m)
```

```
## 161/161 - 2s - loss: 0.3210 - accuracy: 0.9202 - 2s/epoch - 14ms/step
```

```
cat("Test loss:", score_m[[1]], "\n")
```

```
## Test loss: 0.3209823
```

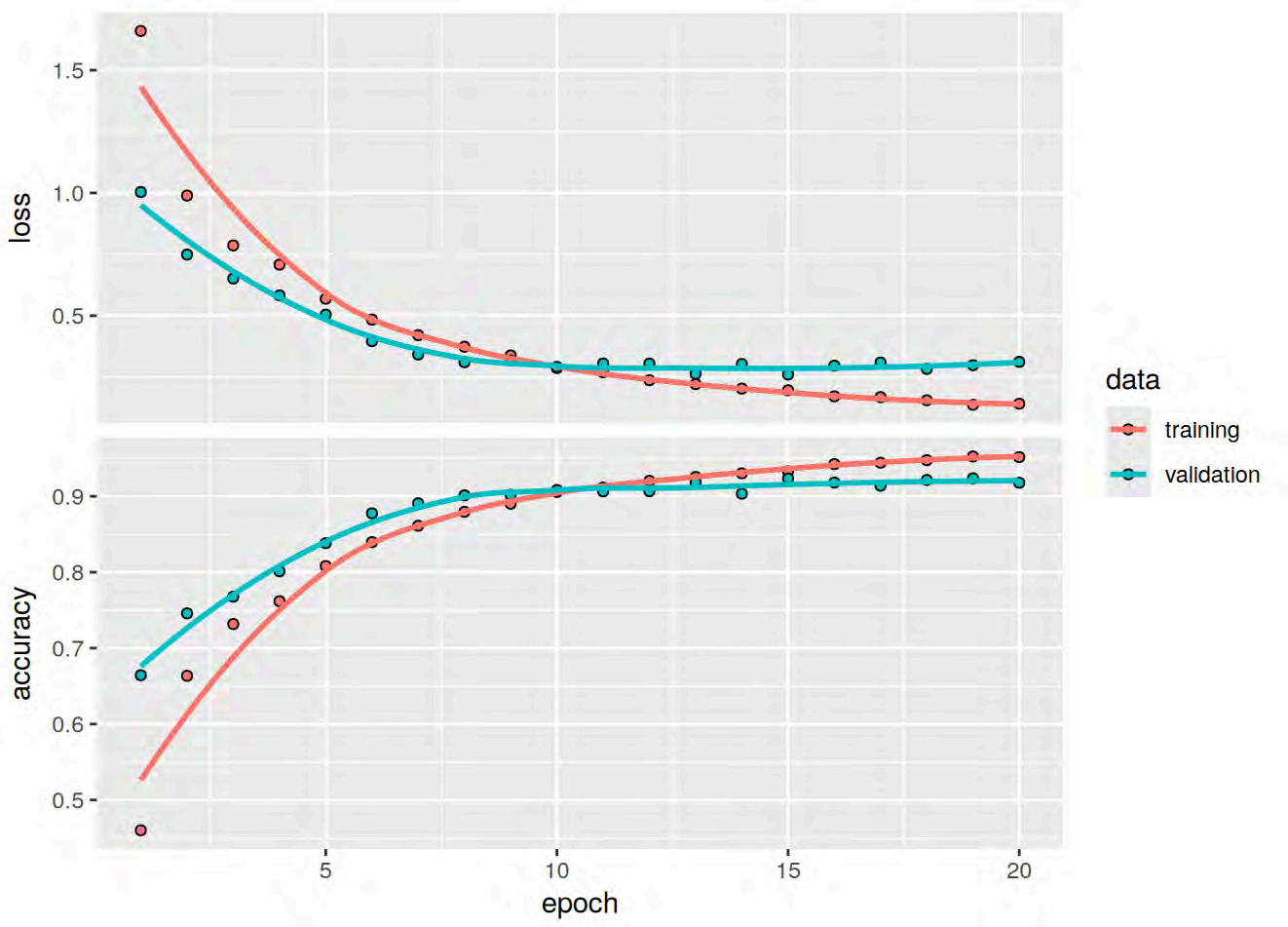
```
cat("Test accuracy:", score_m[[2]], "\n")
```

```
## Test accuracy: 0.920164
```

```
y_prob_m <- model_m %>% predict(x_test_m)
```

```
## 161/161 - 2s - 2s/epoch - 14ms/step
```

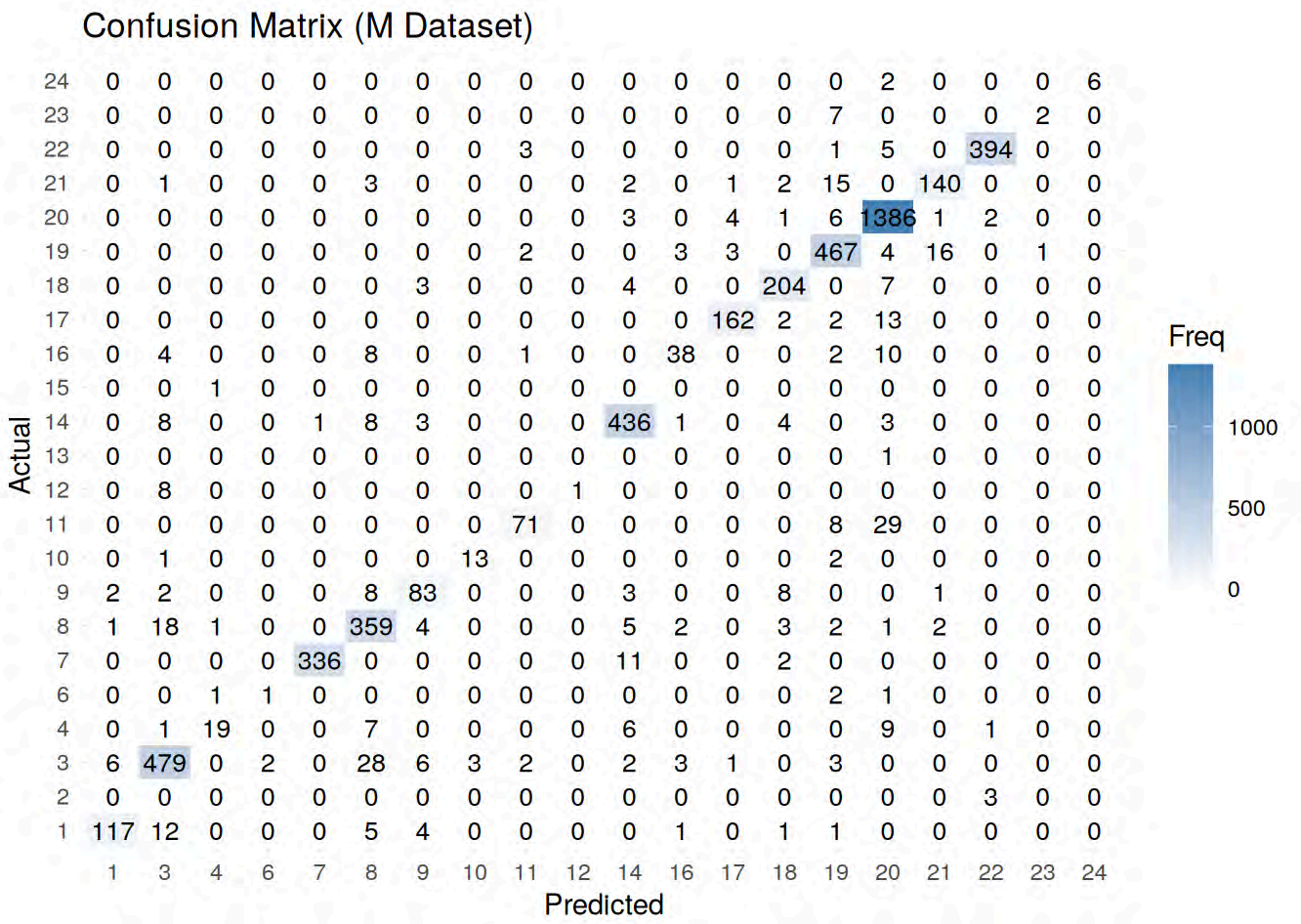
```
y_pred_m <- apply(y_prob_m, 1, which.max)
y_true_m <- apply(y_test_m, 1, which.max)
plot(history_m)
```



Visual Confusion Matrix

```
cm_m <- table(True = y_true_m, Predicted = y_pred_m)
cm_df_m <- as.data.frame(cm_m)
colnames(cm_df_m) <- c("True", "Predicted", "Freq")

ggplot(data = cm_df_m, aes(x = Predicted, y = True, fill = Freq)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "white", high = "steelblue") +
  geom_text(aes(label = Freq), size = 3.5) +
  labs(title = "Confusion Matrix (M Dataset)", x = "Predicted", y = "Actual") +
  theme_minimal()
```



S side

Set directory and load data

```
base_dir_s <- "/UsersLcl/leemin/s side"
class_dirs_s <- list.dirs(base_dir_s, full.names = TRUE, recursive = FALSE)

img_list_s <- list()
label_list_s <- c()

img_to_array_s <- function(img) {
  img <- image_resize(img, "100x100!")
  img_data <- as.integer(image_data(img, channels = "rgb")) / 255
  array(as.numeric(img_data), dim = dim(img_data))
}

for (i in seq_along(class_dirs_s)) {
  class_path <- class_dirs_s[i]
  class_label <- basename(class_path)
  image_files <- list.files(class_path, pattern = "\\.png$", full.names = TRUE)

  for (f in image_files) {
    try({
      img <- image_read(f)
      arr <- img_to_array_s(img)
      img_list_s[[length(img_list_s) + 1]] <- arr
      label_list_s <- c(label_list_s, class_label)
    }, silent = TRUE)
  }
}
```

Prepare data

```
num_s <- length(img_list_s)
img_array_s <- array(0, dim = c(num_s, 100, 100, 3))

for (i in 1:num_s) {
  img_array_s[i,,,] <- img_list_s[[i]]
}

labels_s <- as.factor(label_list_s)
y_s <- to_categorical(as.integer(labels_s) - 1)
n_class_s <- length(levels(labels_s))
```

Split Test and Train

```
set.seed(42)
idx_s <- sample(1:num_s)
train_idx_s <- idx_s[1:floor(0.8 * num_s)]
test_idx_s  <- idx_s[(floor(0.8 * num_s) + 1):num_s]
x_train_s <- img_array_s[train_idx_s,,,]
y_train_s <- y_s[train_idx_s,]
x_test_s <- img_array_s[test_idx_s,,,]
y_test_s <- y_s[test_idx_s,]
```

Define CNN model

```
model_s <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu', input_shape = c(100, 10
0, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_flatten() %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(0.5) %>%
  layer_dense(units = n_class_s, activation = 'softmax')

model_s %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = 'accuracy'
)
```

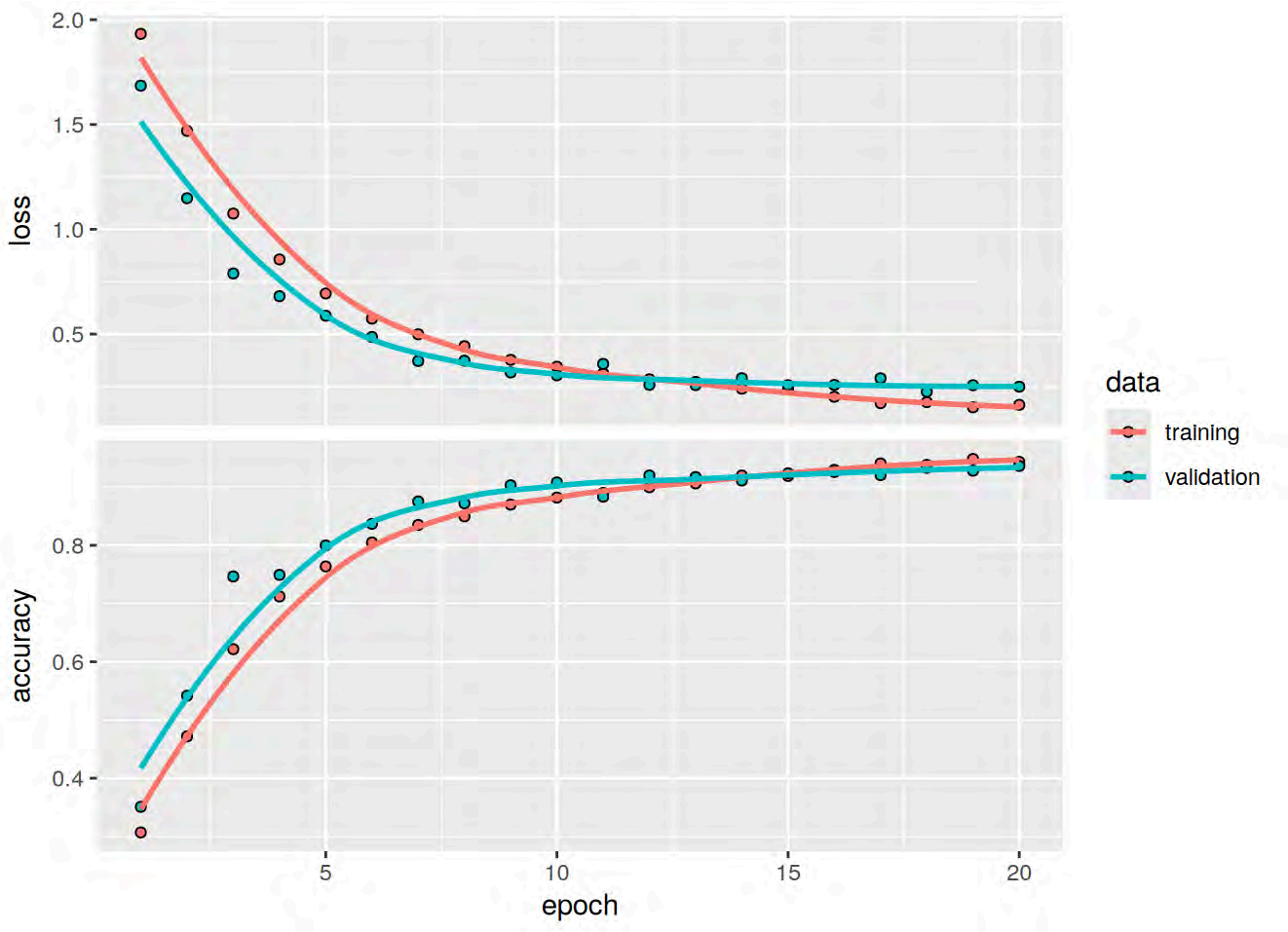
Train Model

```
history_s <- model_s %>% fit(
  x_train_s, y_train_s,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
```

```
## Epoch 1/20
## 230/230 - 15s - loss: 1.9323 - accuracy: 0.3068 - val_loss: 1.6853 - val_accuracy: 0.3507 - 1
5s/epoch - 67ms/step
## Epoch 2/20
## 230/230 - 14s - loss: 1.4694 - accuracy: 0.4718 - val_loss: 1.1482 - val_accuracy: 0.5416 - 1
4s/epoch - 61ms/step
## Epoch 3/20
## 230/230 - 14s - loss: 1.0751 - accuracy: 0.6217 - val_loss: 0.7891 - val_accuracy: 0.7466 - 1
4s/epoch - 61ms/step
## Epoch 4/20
## 230/230 - 14s - loss: 0.8562 - accuracy: 0.7122 - val_loss: 0.6808 - val_accuracy: 0.7493 - 1
4s/epoch - 61ms/step
## Epoch 5/20
## 230/230 - 14s - loss: 0.6937 - accuracy: 0.7638 - val_loss: 0.5875 - val_accuracy: 0.7999 - 1
4s/epoch - 61ms/step
## Epoch 6/20
## 230/230 - 14s - loss: 0.5737 - accuracy: 0.8050 - val_loss: 0.4862 - val_accuracy: 0.8369 - 1
4s/epoch - 61ms/step
## Epoch 7/20
## 230/230 - 14s - loss: 0.4988 - accuracy: 0.8350 - val_loss: 0.3715 - val_accuracy: 0.8755 - 1
4s/epoch - 61ms/step
## Epoch 8/20
## 230/230 - 14s - loss: 0.4420 - accuracy: 0.8499 - val_loss: 0.3729 - val_accuracy: 0.8722 - 1
4s/epoch - 61ms/step
## Epoch 9/20
## 230/230 - 14s - loss: 0.3770 - accuracy: 0.8700 - val_loss: 0.3169 - val_accuracy: 0.9032 - 1
4s/epoch - 61ms/step
## Epoch 10/20
## 230/230 - 14s - loss: 0.3443 - accuracy: 0.8821 - val_loss: 0.3030 - val_accuracy: 0.9081 - 1
4s/epoch - 61ms/step
## Epoch 11/20
## 230/230 - 14s - loss: 0.3123 - accuracy: 0.8901 - val_loss: 0.3576 - val_accuracy: 0.8836 - 1
4s/epoch - 61ms/step
## Epoch 12/20
## 230/230 - 14s - loss: 0.2842 - accuracy: 0.8998 - val_loss: 0.2576 - val_accuracy: 0.9201 - 1
4s/epoch - 61ms/step
## Epoch 13/20
## 230/230 - 14s - loss: 0.2720 - accuracy: 0.9064 - val_loss: 0.2566 - val_accuracy: 0.9173 - 1
4s/epoch - 61ms/step
## Epoch 14/20
## 230/230 - 14s - loss: 0.2393 - accuracy: 0.9199 - val_loss: 0.2894 - val_accuracy: 0.9114 - 1
4s/epoch - 61ms/step
## Epoch 15/20
## 230/230 - 14s - loss: 0.2372 - accuracy: 0.9192 - val_loss: 0.2562 - val_accuracy: 0.9239 - 1
4s/epoch - 61ms/step
## Epoch 16/20
## 230/230 - 14s - loss: 0.2007 - accuracy: 0.9297 - val_loss: 0.2574 - val_accuracy: 0.9260 - 1
4s/epoch - 61ms/step
## Epoch 17/20
## 230/230 - 14s - loss: 0.1702 - accuracy: 0.9407 - val_loss: 0.2889 - val_accuracy: 0.9206 - 1
4s/epoch - 61ms/step
## Epoch 18/20
```

```
## 230/230 - 14s - loss: 0.1754 - accuracy: 0.9383 - val_loss: 0.2246 - val_accuracy: 0.9331 - 1
4s/epoch - 61ms/step
## Epoch 19/20
## 230/230 - 14s - loss: 0.1503 - accuracy: 0.9485 - val_loss: 0.2560 - val_accuracy: 0.9282 - 1
4s/epoch - 61ms/step
## Epoch 20/20
## 230/230 - 14s - loss: 0.1624 - accuracy: 0.9434 - val_loss: 0.2490 - val_accuracy: 0.9364 - 1
4s/epoch - 61ms/step
```

```
plot(history_s)
```



Evaluate

```
score_s <- model_s %>% evaluate(x_test_s, y_test_s)
```

```
## 72/72 - 1s - loss: 0.2667 - accuracy: 0.9365 - 948ms/epoch - 13ms/step
```

```
cat("Test loss:", score_s[[1]], "\n")
```

```
## Test loss: 0.2666971
```

```
cat("Test accuracy:", score_s[[2]], "\n")
```

```
## Test accuracy: 0.9364665
```

```
y_prob_s <- model_s %>% predict(x_test_s)
```

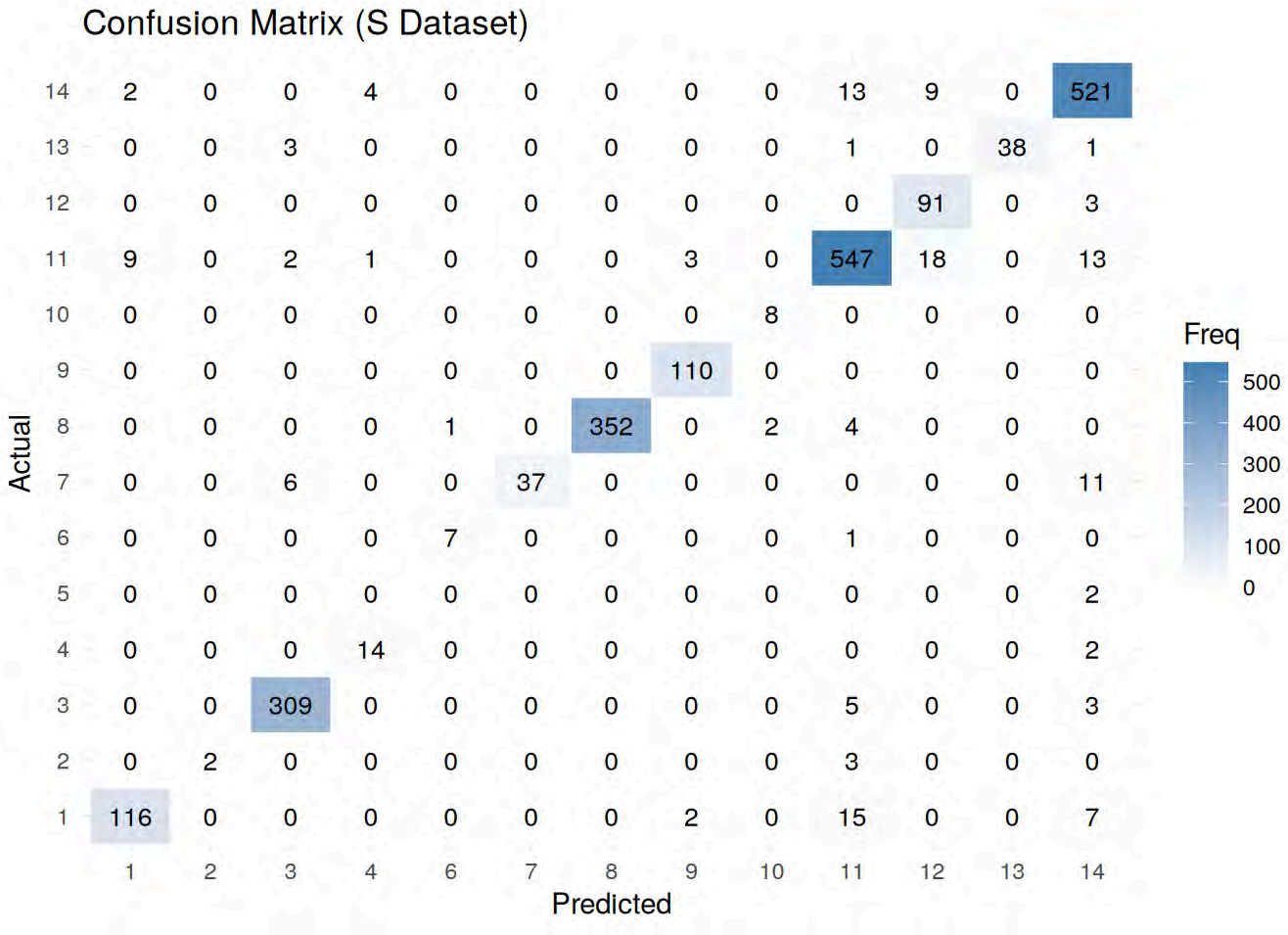
```
## 72/72 - 1s - 976ms/epoch - 14ms/step
```

```
y_pred_s <- apply(y_prob_s, 1, which.max)
y_true_s <- apply(y_test_s, 1, which.max)
```

Visual Confusion Matrix

```
cm_s <- table(True = y_true_s, Predicted = y_pred_s)
cm_df_s <- as.data.frame(cm_s)
colnames(cm_df_s) <- c("True", "Predicted", "Freq")

ggplot(data = cm_df_s, aes(x = Predicted, y = True, fill = Freq)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "white", high = "steelblue") +
  geom_text(aes(label = Freq), size = 3.5) +
  labs(title = "Confusion Matrix (S Dataset)", x = "Predicted", y = "Actual") +
  theme_minimal()
```



Try with M side and S side combined

Set directory and load data

```
base_dir_combined <- "/UsersLcl/leemin/sample"
class_dirs_combined <- list.dirs(base_dir_combined, full.names = TRUE, recursive = FALSE)

img_list_combined <- list()
label_list_combined <- c()

img_to_array <- function(img) {
  img <- image_resize(img, "100x100!")
  img_data <- as.integer(image_data(img, channels = "rgb")) / 255
  array(as.numeric(img_data), dim = dim(img_data))
}

for (i in seq_along(class_dirs_combined)) {
  class_path <- class_dirs_combined[i]
  class_label <- paste0("combined_", basename(class_path))
  image_files <- list.files(class_path, pattern = "\\.png$", full.names = TRUE)

  for (f in image_files) {
    try({
      img <- image_read(f)
      arr <- img_to_array(img)
      img_list_combined[[length(img_list_combined) + 1]] <- arr
      label_list_combined <- c(label_list_combined, class_label)
    }, silent = TRUE)
  }
}
```

Prepare data

```
num_combined <- length(img_list_combined)
img_array_combined <- array(0, dim = c(num_combined, 100, 100, 3))
for (i in 1:num_combined) {
  img_array_combined[i,,,] <- img_list_combined[[i]]
}

labels_combined <- as.factor(label_list_combined)
y_combined <- to_categorical(as.integer(labels_combined) - 1)
n_class_combined <- length(levels(labels_combined))
```

split train and test

```
set.seed(42)
idx_combined <- sample(1:num_combined)
train_idx_combined <- idx_combined[1:floor(0.8 * num_combined)]
test_idx_combined <- idx_combined[(floor(0.8 * num_combined) + 1):num_combined]

x_train_combined <- img_array_combined[train_idx_combined,,,]
y_train_combined <- y_combined[train_idx_combined,]

x_test_combined <- img_array_combined[test_idx_combined,,,]
y_test_combined <- y_combined[test_idx_combined,]
```

Define CNN model

```
model_combined <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu', input_shape = c(100, 10
0, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_flatten() %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(0.5) %>%
  layer_dense(units = n_class_combined, activation = 'softmax')

model_combined %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = 'accuracy'
)
```

train model

```
history_combined <- model_combined %>% fit(
  x_train_combined, y_train_combined,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
```

```
## Epoch 1/20
## 761/761 - 48s - loss: 1.9696 - accuracy: 0.3593 - val_loss: 1.2875 - val_accuracy: 0.5701 - 4
8s/epoch - 63ms/step
## Epoch 2/20
## 761/761 - 47s - loss: 1.2002 - accuracy: 0.6057 - val_loss: 0.8837 - val_accuracy: 0.6953 - 4
7s/epoch - 61ms/step
## Epoch 3/20
## 761/761 - 46s - loss: 0.9006 - accuracy: 0.7012 - val_loss: 0.7218 - val_accuracy: 0.7599 - 4
6s/epoch - 61ms/step
## Epoch 4/20
## 761/761 - 46s - loss: 0.6720 - accuracy: 0.7780 - val_loss: 0.5228 - val_accuracy: 0.8309 - 4
6s/epoch - 61ms/step
## Epoch 5/20
## 761/761 - 47s - loss: 0.5301 - accuracy: 0.8280 - val_loss: 0.4806 - val_accuracy: 0.8518 - 4
7s/epoch - 61ms/step
## Epoch 6/20
## 761/761 - 46s - loss: 0.4444 - accuracy: 0.8543 - val_loss: 0.4123 - val_accuracy: 0.8654 - 4
6s/epoch - 61ms/step
## Epoch 7/20
## 761/761 - 46s - loss: 0.3892 - accuracy: 0.8728 - val_loss: 0.3605 - val_accuracy: 0.8846 - 4
6s/epoch - 61ms/step
## Epoch 8/20
## 761/761 - 47s - loss: 0.3440 - accuracy: 0.8868 - val_loss: 0.3371 - val_accuracy: 0.8970 - 4
7s/epoch - 62ms/step
## Epoch 9/20
## 761/761 - 46s - loss: 0.3070 - accuracy: 0.8979 - val_loss: 0.2953 - val_accuracy: 0.9134 - 4
6s/epoch - 61ms/step
## Epoch 10/20
## 761/761 - 46s - loss: 0.2743 - accuracy: 0.9078 - val_loss: 0.3211 - val_accuracy: 0.8993 - 4
6s/epoch - 61ms/step
## Epoch 11/20
## 761/761 - 46s - loss: 0.2385 - accuracy: 0.9238 - val_loss: 0.2966 - val_accuracy: 0.9132 - 4
6s/epoch - 61ms/step
## Epoch 12/20
## 761/761 - 46s - loss: 0.2187 - accuracy: 0.9260 - val_loss: 0.2878 - val_accuracy: 0.9165 - 4
6s/epoch - 61ms/step
## Epoch 13/20
## 761/761 - 46s - loss: 0.2127 - accuracy: 0.9283 - val_loss: 0.3115 - val_accuracy: 0.9162 - 4
6s/epoch - 61ms/step
## Epoch 14/20
## 761/761 - 46s - loss: 0.1947 - accuracy: 0.9339 - val_loss: 0.2748 - val_accuracy: 0.9236 - 4
6s/epoch - 61ms/step
## Epoch 15/20
## 761/761 - 46s - loss: 0.1676 - accuracy: 0.9435 - val_loss: 0.3062 - val_accuracy: 0.9201 - 4
6s/epoch - 61ms/step
## Epoch 16/20
## 761/761 - 46s - loss: 0.1663 - accuracy: 0.9428 - val_loss: 0.3417 - val_accuracy: 0.9164 - 4
6s/epoch - 61ms/step
## Epoch 17/20
## 761/761 - 46s - loss: 0.1485 - accuracy: 0.9493 - val_loss: 0.3086 - val_accuracy: 0.9259 - 4
6s/epoch - 61ms/step
## Epoch 18/20
```

```
## 761/761 - 46s - loss: 0.1401 - accuracy: 0.9505 - val_loss: 0.3185 - val_accuracy: 0.9251 - 4
6s/epoch - 61ms/step
## Epoch 19/20
## 761/761 - 46s - loss: 0.1348 - accuracy: 0.9543 - val_loss: 0.3002 - val_accuracy: 0.9329 - 4
6s/epoch - 61ms/step
## Epoch 20/20
## 761/761 - 46s - loss: 0.1234 - accuracy: 0.9575 - val_loss: 0.3273 - val_accuracy: 0.9208 - 4
6s/epoch - 61ms/step
```

evaluate

```
score_combined <- model_combined %>% evaluate(x_test_combined, y_test_combined)
```

```
## 238/238 - 3s - loss: 0.3476 - accuracy: 0.9206 - 3s/epoch - 14ms/step
```

```
cat("Test loss:", score_combined[[1]], "\n")
```

```
## Test loss: 0.3475748
```

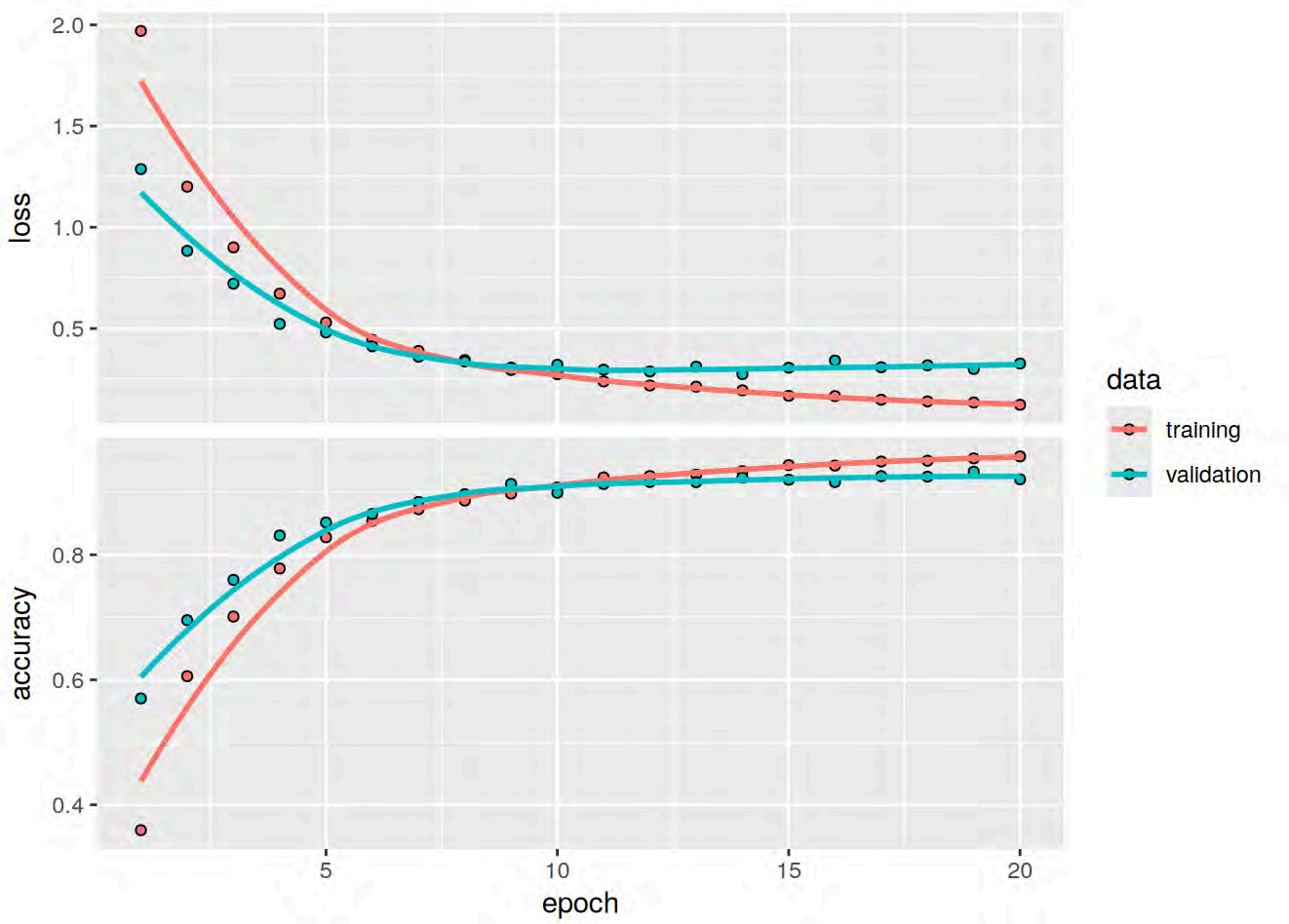
```
cat("Test accuracy:", score_combined[[2]], "\n")
```

```
## Test accuracy: 0.920589
```

```
y_prob_combined <- model_combined %>% predict(x_test_combined)
```

```
## 238/238 - 3s - 3s/epoch - 13ms/step
```

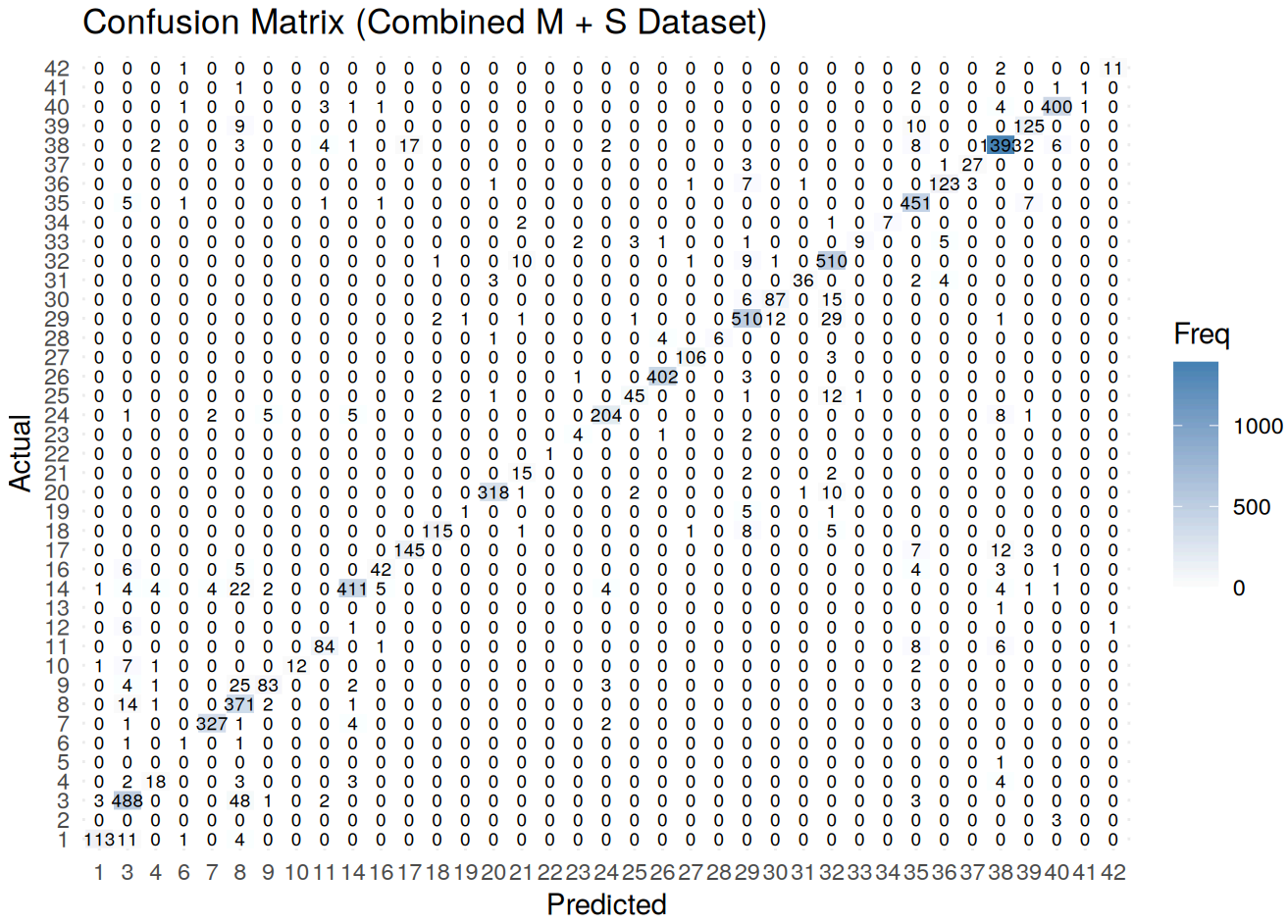
```
y_pred_combined <- apply(y_prob_combined, 1, which.max)
y_true_combined <- apply(y_test_combined, 1, which.max)
plot(history_combined)
```



Visual confusion matrix

```
cm_combined <- table(True = y_true_combined, Predicted = y_pred_combined)
cm_df_combined <- as.data.frame(cm_combined)
colnames(cm_df_combined) <- c("True", "Predicted", "Freq")

ggplot(data = cm_df_combined, aes(x = Predicted, y = True, fill = Freq)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "white", high = "steelblue") +
  geom_text(aes(label = Freq), size = 2.5) +
  labs(title = "Confusion Matrix (Combined M + S Dataset)", x = "Predicted", y = "Actual") +
  theme_minimal()
```


Confusion Matrix (Combined M + S Dataset)
Actual
Predicted
Freq
1000
500
0