Neural Network Write Up

Alex Hey

December 1, 2024

Objective

Today's goal was to preprocess a **thinned dataset** of image data (limited to 100 images per category), train a neural network using Keras, and evaluate its performance. The dataset consisted of three categories: **cats**, **dogs**, and **snakes**.

Dataset Preparation for Day01

To reduce complexity:

- Only 100 images per category were loaded.
- Labels were assigned to correspond to categories: cats (0), dogs (1), and snakes (2).
- For Day 2, 500 images were used to thin the dataset

Day 1: Initial Model Training

Original Model

The first model used the `sparse_categorical_crossentropy` loss function with a dropout rate of 0.5. Below is the accuracy plot:

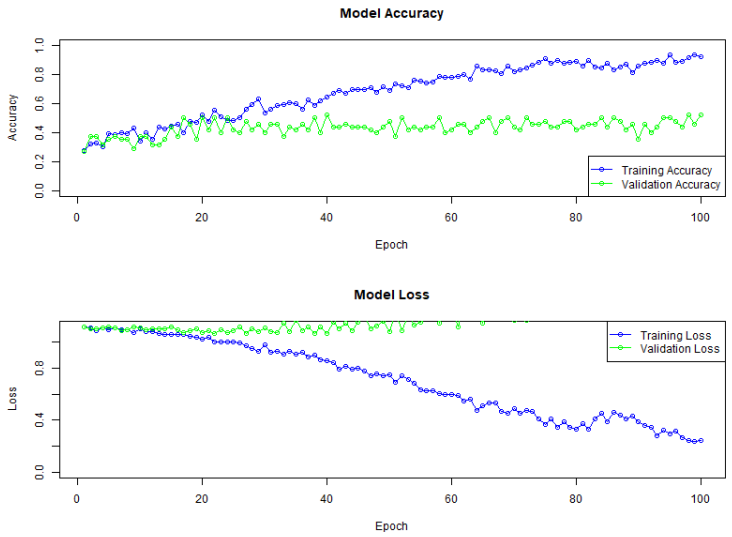

Figure 1: Day 1: Original Model Accuracy Over Epochs (Sparse Categorical Crossentropy)

This model did not perform well. Validation accuracy never got to a point where it wa smuch better than a random guess, which would be about 30 percent accuracy. Further tuning of dropout rates or experimenting with a different loss function might improve performance.

Day 2: Modified Models

Model 1: Reduced Dropout Rate

This model decreased the dropout rate from 0.5 to 0.3,
while keeping the loss function as `sparse_categorical_crossentropy`. Below is the accuracy plot:

This adjustment was made to test if a lower dropout rate would reduce overfitting. I also altered the image processing before to do 500 images per category.

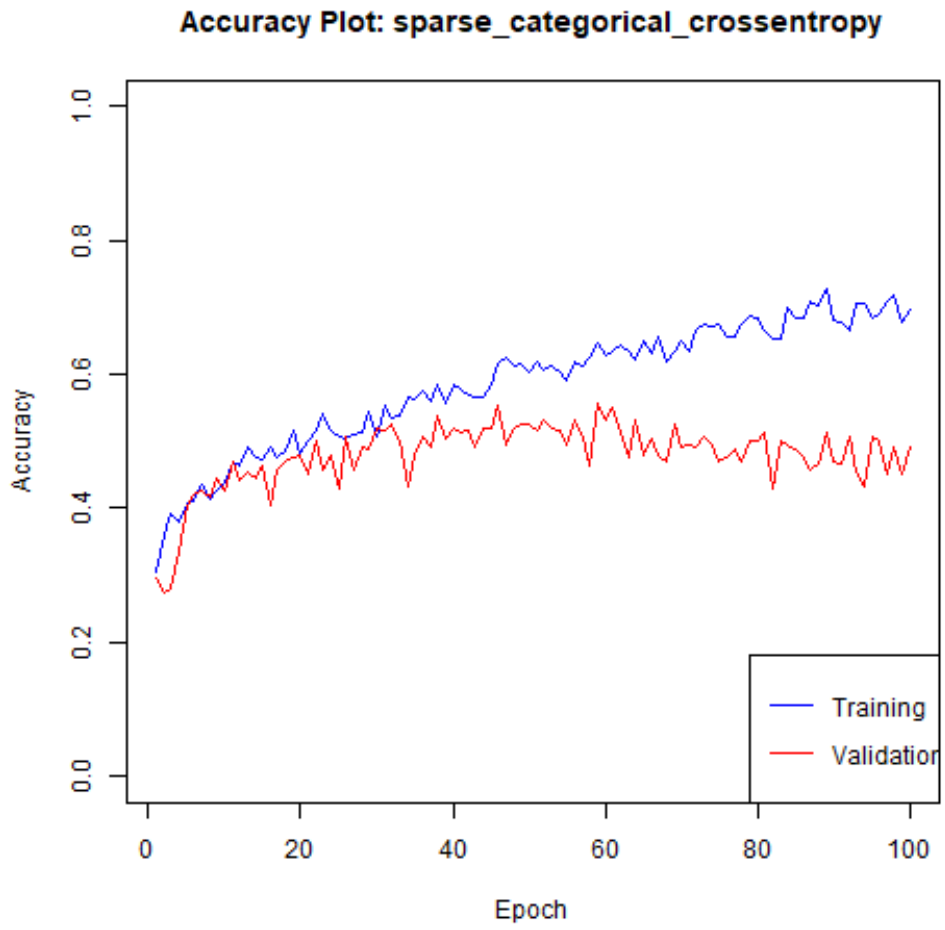

Figure 2: Model 1: Reduced Dropout Rate Accuracy Over Epochs (Sparse Categorical Crossentropy)

This model showed some improvement in training accuracy but struggled with validation accuracy with that only improving slightly from the previous model. The next model will change the loss function to see if that helps:

Model 2: One-Hot Encoding and `categorical_crossentropy`

This model used the `categorical_crossentropy` loss function and required one-hot encoded labels. Below is the accuracy plot:

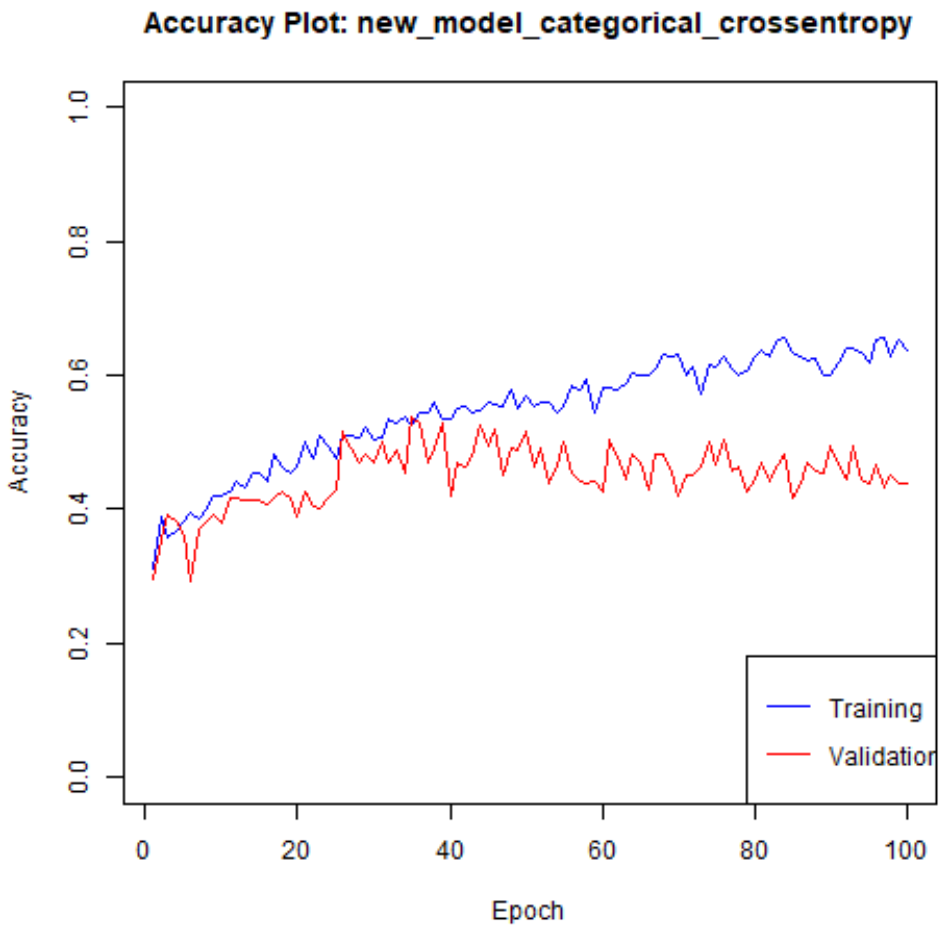

Figure 3: Model 2: Categorical Crossentropy Accuracy Over Epochs

The results were slightly better than the reduced dropout rate model but still indicated issues with generalization.

Discussion and Observations

Comparison of Models Summary

1. **Original Model (Day 1)**: - Showed steady improvement in training accuracy over 100 epochs. - Validation accuracy was erratic, indicating overfitting.

2. **Model 1 (Reduced Dropout Rate)**: - Slight improvement in training accuracy and validation accuracy. Not where I want it to be

3. **Model 2 (`categorical_crossentropy`)**: - Changing the loss function did not lead to an improvement

Key Observations

The models performed poorly overall. Below are examples of correctly classified and misclassified images, I don't see a reason why this model did not get those right. Moving forward, I plan to explore image preprocessing techniques, as observed in Dan's work with the same dataset for CNNs, to improve results.

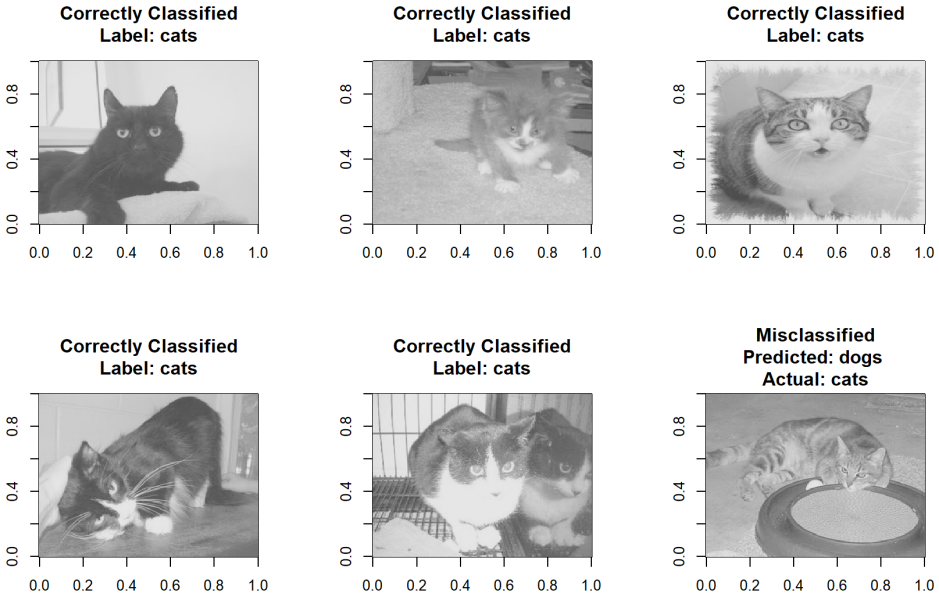

Figure 4: Examples of Correctly Classified Images

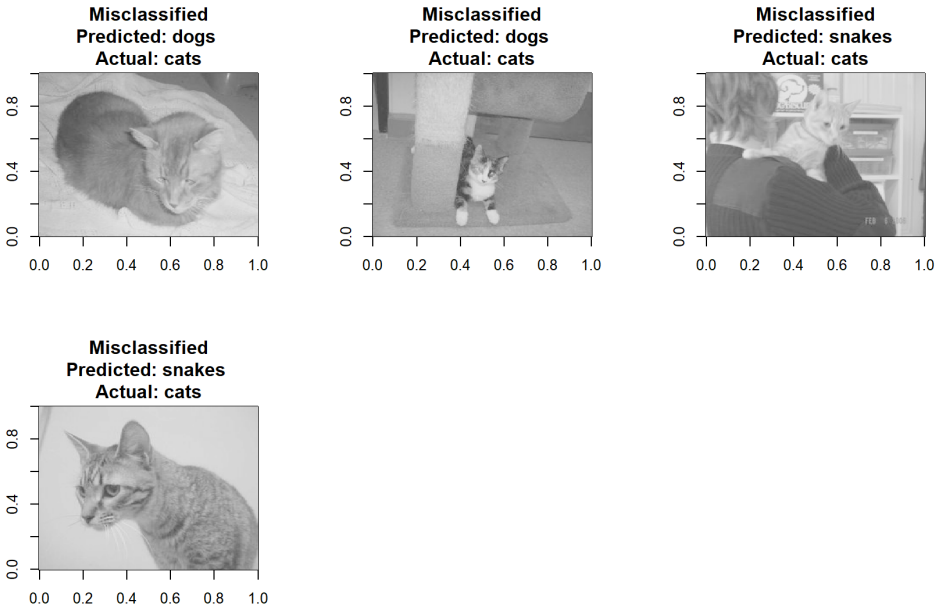

Figure 5: Examples of Misclassified Images

1 Day 03 and 4: Convolutional Network

I processed the data in the way I saw you do it in class. I think this helped a lot, as it kept the images in color. I got way better perfromance with the below models:

1.1 Model 1: Simple CNN

Model 1 is the simplest of the three architectures, designed as a baseline. The code for Model 1 is shown below:

```
model_1 <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = 'relu', input_shape = c(256, 256, 3))
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 3, activation = 'softmax')

model_1 %>% compile(
  optimizer = 'adam',
  loss = 'categorical_crossentropy',
  metrics = c('accuracy')
)
```

This model uses the `Adam` optimizer.

1.2 Model 2: Larger CNN with Additional Layers

Model 2 introduces more complexity with more layers:

```
model_2 <- keras_model_sequential() %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = 'relu', input_shape = c(256, 256, 3))
  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_dense(units = 3, activation = 'softmax')

model_2 %>% compile(
  optimizer = 'sgd',
  loss = 'categorical_crossentropy',
  metrics = c('accuracy')
)
```

This model uses the `SGD` (Stochastic Gradient Descent) optimizer, which "updates weights incrementally and is known for its stability."

1.3 Model 3: CNN with Dropout Regularization

Model 3 incorporates dropout layers for regularization:

```
model_3 <- keras_model_sequential() %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = 'relu', input_shape = c(256, 256, 3))
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.25) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = 'relu') %>%
```

```
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_flatten() %>%
  layer_dense(units = 512, activation = 'relu') %>%
  layer_dense(units = 3, activation = 'softmax')

model_3 %>% compile(
  optimizer = 'rmsprop',
  loss = 'categorical_crossentropy',
  metrics = c('accuracy')
)
```

This model utilizes the `RMSprop` optimizer, which "divides the learning rate by a moving average of the magnitudes of recent gradients."

1.4 Model 1 Performance

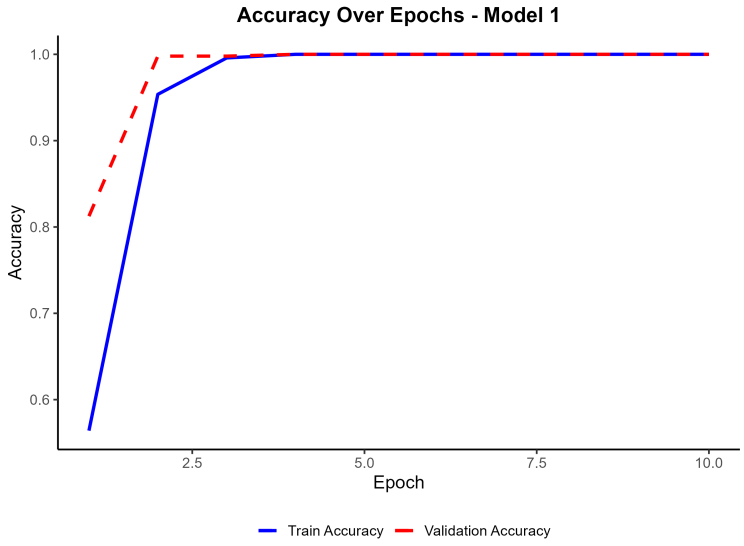

Figure 6: Accuracy trends for Model 1. Train accuracy is shown in blue, and validation accuracy is shown in red.

1.5 Model 2 Performance

Model 2 also showed improvements in training accuracy but experienced significant fluctuations in validation accuracy. I'm not sure why it is such a sharp increase and decrease in validation accuracy, but it eventually stabilizes!

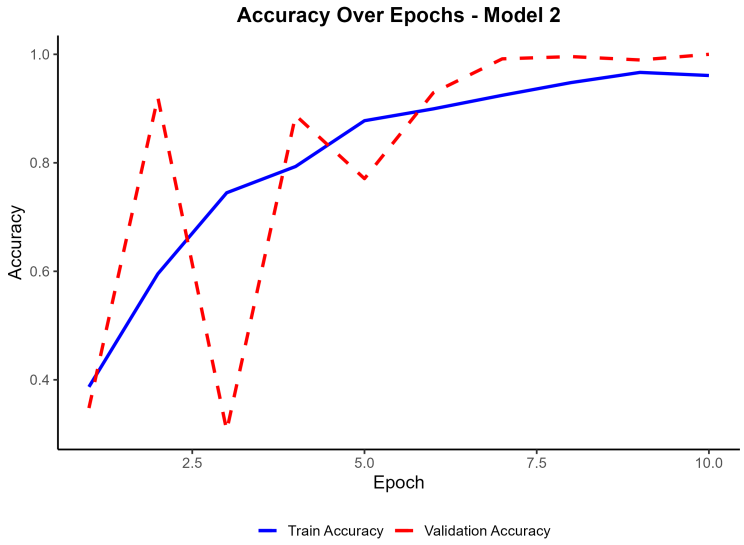

Figure 7: Accuracy trends for Model 2. Train accuracy is shown in blue, and validation accuracy is shown in red.

1.6 Model 3 Performance

Model 3 underperformed compared to the other models. Both train and validation accuracies remained low, indicating strong regularization effects due to the dropout layers, possibly leading to underfitting. I also think if I ran this for more epochs it could have done better, but still it likely wouldn't outperform model 1.

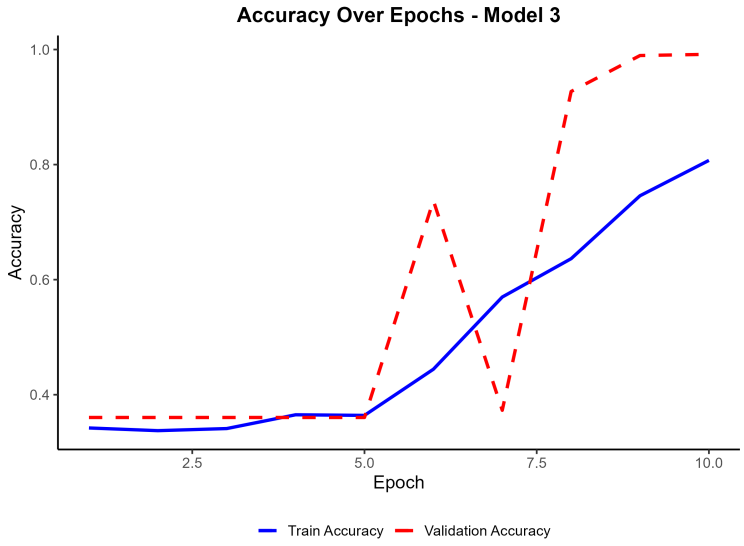

Figure 8: Accuracy trends for Model 3. Train accuracy is shown in blue, and validation accuracy is shown in red.

Table 1: Test Accuracy of Different Models from Day 3-4

Model	Test Accuracy
Model 1 (Simple CNN)	1.000
Model 2 (Larger CNN)	0.748
Model 3 (CNN with Dropout)	0.998

It looks like Model 1 is the best. it correctly predicted all of the testing data.

2 Day 5: Transfer Learning

The pre-trained MobileNetV2 model was used as the base feature extractor, followed by a custom architecture similar to the previously best-performing model. The following code demonstrates the setup of the transfer learning model:

```
# Load MobileNetV2 pre-trained model without top layers
base_model <- application_mobilenet_v2(
  include_top = FALSE,          #Exclude original classifier
  weights = "imagenet",         #pre-trained ImageNet weights
  input_shape = c(256, 256, 3)  #match input shape from before
)

#cool freeze weights function
freeze_weights(base_model)

#Build the transfer learning model
transfer_model <- keras_model_sequential() %>%
  base_model %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 3, activation = 'softmax')

#compile
transfer_model %>% compile(
  optimizer = 'adam',
  loss = 'categorical_crossentropy',
  metrics = c('accuracy')
)
```

3 Training and Evaluation

The transfer learning model was trained using the same training and testing datasets as in Day 3. The training process spanned 10 epochs, with a batch size of 32:

```
# Train the model
history_transfer <- transfer_model %>% fit(
  x = x_train,
  y = y_train,
  epochs = 10,
  batch_size = 32,
  validation_data = list(x_test, y_test)
```

```
)

#Evaluate the model
test_eval <- transfer_model %>% evaluate(x_test, y_test)
```

The transfer model's accuracy was compared to the previous best-performing model's accuracy from Day 3 and 4 (Model 1). This was the most simple model, and its' accuracy could not really be beat.

4 Results

4.1 Accuracy Plot

The training and validation accuracy trends for the transfer learning model are shown in Figure 9.

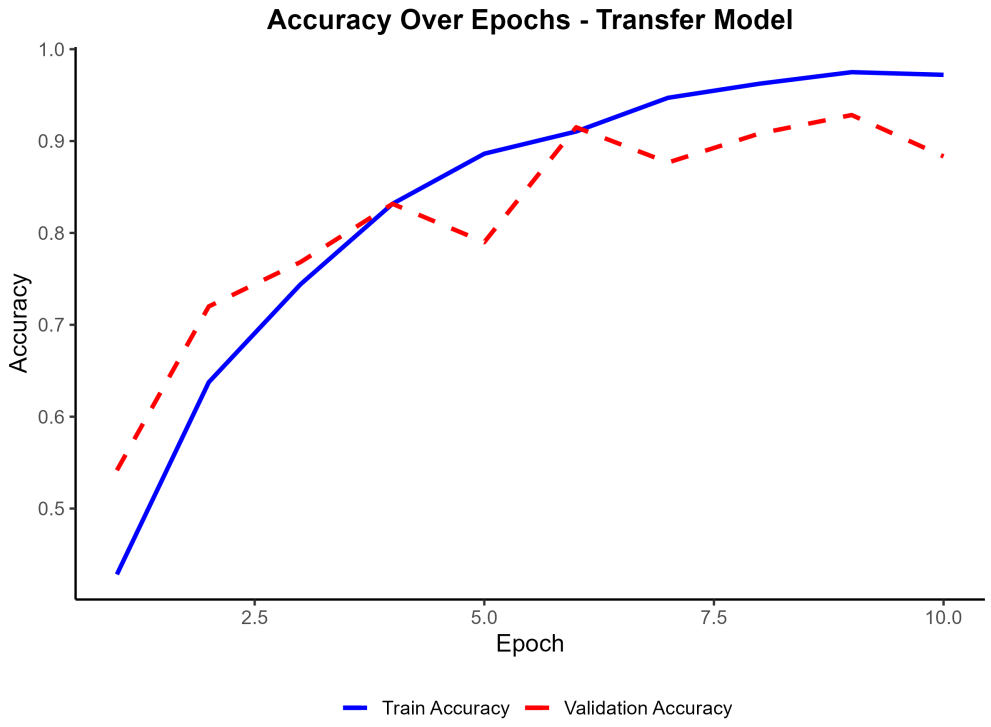

Figure 9: Accuracy trends for the transfer learning model. Train accuracy is shown in blue, and validation accuracy is shown in red.

4.2 Accuracy Comparison

Table 2 compares the test accuracy of the transfer learning model with the best-performing model (Model 1) from Day 3. I don't think I am surprised by this, as the more complicated models the other day did not do as well as the simple one, so these pretrained models online were to complex for this "simple" classification problem. The simple model was probably more effective for this specific dataset because it had smaller and more task-oriented architecture that would have allowed it to focus on learning the problem features directly without having to first relearn pre-trained weights?

Model	**Test Accuracy**
Transfer Learning Model	0.91667
Best Model (Day 3)	1.0000

Table 2: comparisons