Sorting Recyclable Materials with Neural Networks

DSP 566 — Ryan Soucy and Nellie Dawson

ABSTRACT:

In order to assist in combating the contamination in recycled material sorting, we build a series of neural networks to classify trash images into six classes using a dataset provided by Stanford students. By using Convolutional Neural Networks and pretrained Resnet50 models, we explore regularizer and optimizer parameters to improve upon initial model accuracies. We see quite varied and unstable results in terms of accuracy and loss as well as signs of early overfitting throughout the documented model building. As a model is only as good as the training data being fed into it, we consider that the quality of the dataset's images may have some impact on the difficulties in achieving accurate models despite the parameter testing and tuning done.

INTRODUCTION

Landfills across the world are filling up while many recyclable materials are often added to these landfills due to negligence or contamination. In other parts of the world, namely Japan, waste removal comes every day for a different type of recyclable or trash material. These systems are called multi-stream recycling efforts, and America has only made the change to single-stream efforts over the last decade or so. Single-stream recycling is a convenient way for individuals to send their recyclable materials to facilities that do the sorting for them. However, these products need to be separated at the facility to go through the recycling process and these facilities often use “rudimentary methods” such as using magnets, air pressure, gravity, etc. Although single stream recycle collection has increased recycling percentages, it has put strain on the facilities’ need to sort the products and has simultaneously increased the percentage of contamination in recycling. With the increase in materials in these sorting facilities, workers are overwhelmed by the amount of waste and struggle to effectively reduce contamination of recyclable materials. Data suggest that only approximately 5-10% of plastic goes through the complete recycling process, and the rest continues to landfills and incinerators. (Rivero, 2024). It can be said that the percentage increase of recycling material outweighs the increase in contamination percentage, but it is clear there is room for improvement in these sorting practices. With the use of algorithms, we hope to make this sorting process more effective by determining if we can identify the types of recyclable material images and distinguish between them and other trash.

SIMILAR RESEARCH

Two Stanford students, Gary Thung and Mary Yang, built this dataset by taking images of trash against a white background and sorting them into six categories. They used support vector machines (SVM) and convolutional neural networks (CNNs) to sort these images. Their SVM methods saw a 63% test accuracy while the CNNs only saw a test accuracy of 22%. Since submitting their paper, they were able to achieve an approximate 75% test accuracy after utilizing other weight initialization methods.

An organization named GreyParrot has been developing waste sorting algorithms and machines since 2019 and has recently partnered with Bollegraaf, a company that has built thousands of recycling plants worldwide including 340 in North America. (Rivero, 2024). They combine their deep learning with machinery to view trash and recyclable materials on a conveyor belt to identify and sort the contents by looking at seven layers of detail: composition, mass, financial value, food-grade and function, size, brand and SKU, emission (GreyParrot, 2019). With more than 100 AI trash spotters in about 50 sorting facilities around the world, GreyParrot and Bollegraaf intend to retrofit their thousands of facilities worldwide to combat the sorting dilemmas these recycling plants face.

METHODS

Through a series of neural networks, including both organically built models and pretrained models, we will attempt to accurately classify waste images into six classes. Our dataset consists of over 2500 labeled images that have all been resized to 512 x 384 pixels. Of these images, 501 are glass, 594 are paper, 403 are cardboard, 482 are plastic, 410 are metal, and 137 are trash. We make subsets of these datasets to have equal numbers of each type of material and in order to be able to maximize photo resolution and still run models in a reasonable amount of time. All models will run on 40 epochs but will implement Early Stopping as it monitors the validation loss. We will ensure the model runs through at least 10 epochs, but then will use a patience of 5 at which it stops the model early when we see 5 increases of validation loss.

We begin by building a simple convolutional neural network with a handful of layers increasing in number of neurons from 32 to 128. In this model we use alternating Convolution2D layers and MaxPooling using the Keras Functional API. In this model we hypertune regularization methods to see which is most successful for the data at hand. The regularization methods we use are Batch Normalization, L1 and L2 individually as well as at the same time. We then explore the impact of the optimizer on the model accuracy. The optimizers we test are Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam), and Root Mean Square Propagation (RMSProp). We document each change and make note of which selection does the best at that point in the analysis. Once we explore these parameter tuning and assess model accuracy, we use this as a comparison to a pretrained model available in keras. We choose the Resnet50 pretrained model with frozen weight and feature extraction. When using Resnet50 we explore the impact of the same three optimizers, SGD, Adam and RMSProp. Because of the large models, image sizes and datasets and devices with limited computing power, we use a GPU to run these models.

RESULTS

As seen in **Table 1**, the original simple CNN only saw an accuracy of about 23.7% when sorting images into 6 categories of glass, paper, cardboard, plastic, metal, and trash. Although this is better than the 16.7% chance of randomly selecting the category accurately, we hoped to see better results by tuning our model. With a goal of finding a model with reasonable accuracy while avoiding overfitting, we found some success in adding regularization methods to our original model. Both

BatchNormalization, Dropout, and the combination of using L1 and L2 regularization techniques showed an increase in accuracy to about 35%. When exploring the impact of optimizers, we saw varied results. Although we saw a 35% accuracy the first time we ran a model with an SGD optimizer and L1L2 regularizer, this second time the exact model was run, we only saw a 24.6% accuracy. This volatility in the model seems to be a result of overfitting and poor data quality. The Adam optimizer saw the best results of the three explored, but did not see higher results with the approximately equivalent 35% accuracy.

Model Type	**Tuning**	**Parameter Selection**	**Test Loss**	**Test Accuracy**
Convolutional Neural Network	Original Model	Conv2D (32, 64, 128) w/ MaxPooling	1.67895	0.2368
	Regularization w/ SGD Optimizer	Batch Normalization	1.6309	0.3596
		L2 (.01)	2.12978	0.25438
		L1 (.01)	4.6101	0.1754
		L1L2 (.01,.01)	4.26664	0.3508
		Dropout	4.99869	0.2982
	Optimizer w/ L1L2 Regularization	SGD	4.2585	0.2456
		Adam	4.9036779	0.350877
		RMSProp	4.712	0.22807
Resnet50	Optimizer w/ Frozen Weights & Feature Extraction	SGD	32.7089	0.69298
		Adam	21.336789	0.6666
		RMSProp	21.7555	0.719298

Table 1 — Model Results

When incorporating the Resnet50 pretrained model, our accuracy rates skyrocketed, but so did our loss values. We froze the weights using imagenet weight initialization and used feature extraction to achieve the most accurate models possible. Here, the RMSProp optimizer performed the best with an accuracy of 71.9% while SGD and Adam saw accuracy levels of 66.6% and 69.2% respectively.

In many models, we saw signs of overfitting throughout the process. Through the implementation of Early Stopping, many models showed signs of overfitting relatively early in the number of epochs. The times we saw the least overfitting were in the models that utilize Stochastic Gradient Descent, but they still stop in a limited number of epochs. Although we experimented with multiple techniques to improve upon this overfitting, we hypothesize that this may be due to the poor image quality.

The image quality amongst the dataset is quite variable. Photos were taken on three different types of cameras and were stated to be taken on a white background with both natural and artificial lighting. Manually combing through the dataset showed variable shades of white background (with some pictures not taken on a white background at all), variable zoom on the objects photographed, and most significantly, variable lighting of the images. Because of this, our model had a significant number of variables at play outside of simply identifying an object within an image. We believe that these variables likely greatly impacted the quality of our results because a model could only be as high quality as the data fed into it.

Below in **Figure 1** is an example plot of training and validation accuracy across epochs as we fit one of our models. This model included both L1 and L2 regularizers to try to reduce overfitting and used the Adam optimizer. We found that the Adam optimizer provided us with higher validation and test set accuracies, but the model still overfit quite early on, even with regularizers. This overfitting trend seemed to occur in almost all of our models, no matter how high the test set accuracy got. This test set accuracy was only about .35.

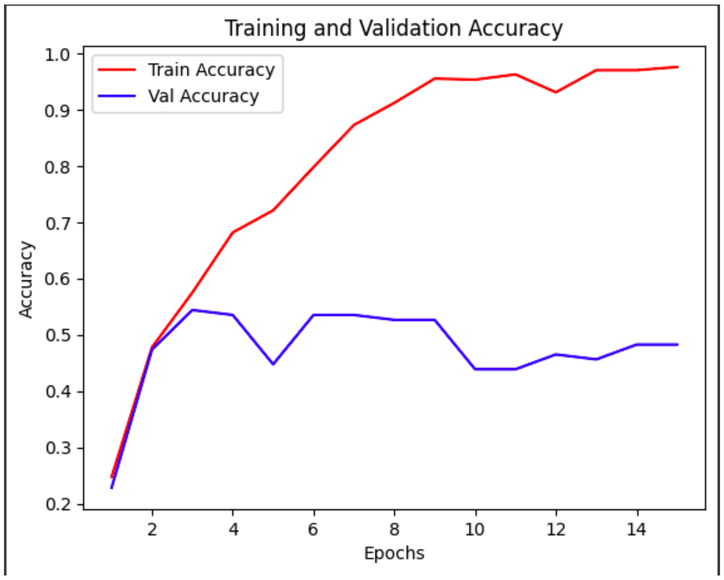


Figure 1 - Training and Validation Accuracy of L1L2 Regularization and Adam Optimizer

Figure 2, below, is an example of one of our best models. This was a pretrained resnet50 model, which achieved a test set accuracy of about .72. You can see that, even using a highly complex pretrained model with frozen weights and achieving a much higher test set accuracy, our model still appears to be very overfit with almost perfect training accuracy and a plateaued validation set accuracy.

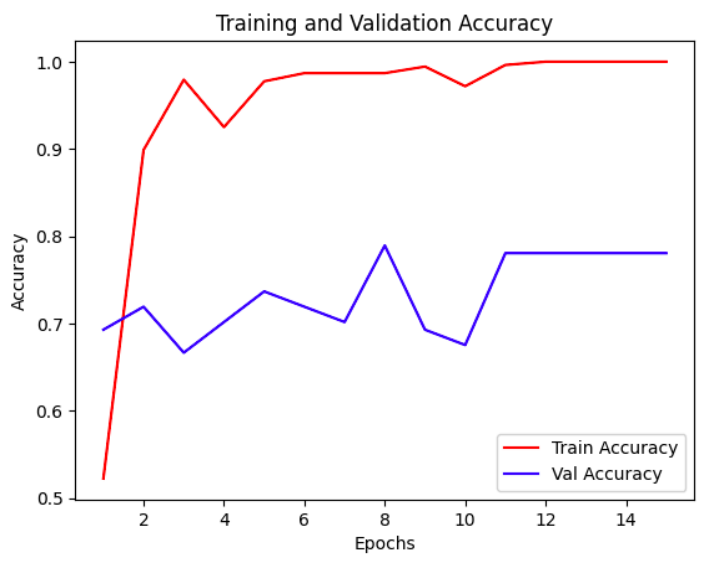


Figure 2 - Training and Validation Accuracy of ResNet50 Pretrained Model

We were able to build some models that did not tend to overfit the training dataset, as is displayed in **Figure 3** below. However, these models often did not overfit because both the training set accuracy and validation set accuracy are incredibly low. The model would not overfit, but this was due to overall low accuracy with a test set accuracy of only .25.

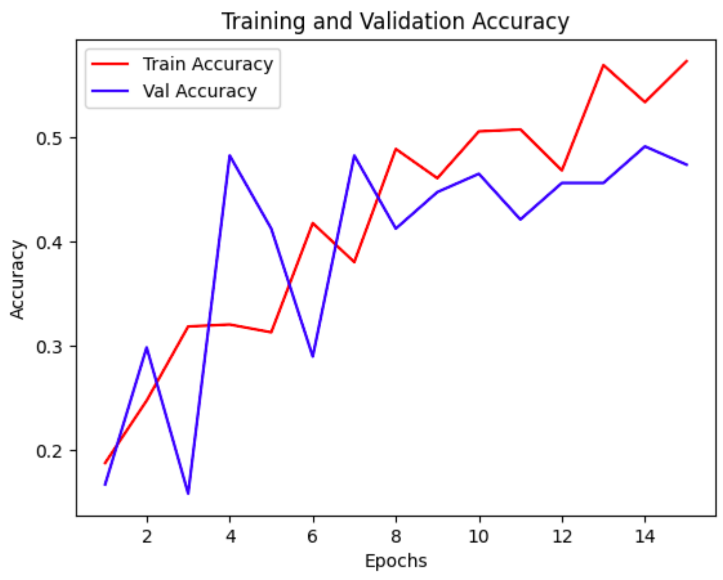


Figure 3 - Training and Validation Accuracy of SGD Optimizer with L1L2 Regularization

CONCLUSION/DISCUSSION

Initially, we saw similarly low accuracy results with our CNN to the Stanford group's CNNs, but were able to achieve higher results by using the pretrained resnet50 model. Our Convolutional Neural Networks achieved accuracy ranges between 17.5% - 35.9%. It seems as though the Stochastic Gradient Descent optimizer showed the least amount of overfitting. The two best models to achieve a 35% accuracy were the Adam Optimizer with L1L2 Regularization and the SGD Optimizer with Batch Normalization. The Stanford students 22% CNN accuracy is within this range. With the variability of our models, it is unclear if our models were consistently more accurate than the Stanford student group.

When incorporating the pretrained Resnet50 model, we achieved better accuracy than the Stanford students' 63% SVM accuracy with our models ranging in accuracy from 66.6% - 71.9%. The optimizer that showed the best results with this pretrained model was the RMSProp Optimizer.

Although it is not clear what kind of deep learning GreyParrot utilizes, they report 95%+ accuracy across 89 categories of material. Our models do not achieve this level of accuracy, but their data collection is much better quality than the images we have access to.

FUTURE CONSIDERATIONS/WORK

The largest goal for future work on this project would be to increase the test set accuracy of our models. We believe that the best way to do this is to obtain a larger and higher quality dataset. If we could build or obtain images of recyclable materials that are less variable, we believe that this would greatly increase the accuracy of our models. To do this, the dataset would have to be collected using the same camera, the same white background, and the same lighting for each image. This will decrease all of the other variables and allow our model to be trained on a high quality dataset where the majority of the variables relate to the recyclable material being photographed.

Once we obtained or built a higher quality dataset, we would continue to tune and improve the models until we obtained highly accurate results. This process could include exploring other model types that we have yet to test. From here, the goal would be to connect this model with engineers who could build machines that could use this sorting algorithm that we have begun to develop. These machines could then be connected to recycling centers to help aid in the sorting process of the recyclable materials, decrease contamination within the recycling process, and decrease the overall costs of recycling.

Sources

Druckman, M., Mitra, A., & Sivacki, N. (2019). *Grey parrot*. Greyparrot Waste Intelligence. https://www.greyparrot.ai/

Rivero, N. (2024, February 7). How the world of recycling is about to be transformed. *The Washington Post*. https://www.washingtonpost.com/climate-solutions/2024/02/07/ai-recycling-sorting/

Scientific American. (2013, September 18). Single-Stream recycling. *Scientific American*. https://www.scientificamerican.com/article/single-stream-recycling/

Thung, G. (2023). *GitHub - Garythung/trashnet: Dataset of images of trash; Torch-based CNN for garbage image classification*. GitHub. https://github.com/garythung/trashnet/

```
import tensorflow as tf
import os, shutil, pathlib
from tensorflow.keras.utils import image_dataset_from_directory
import keras
from keras import layers
from keras.layers import Input, Dense
from keras.models import Model
from keras import regularizers
import matplotlib.pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
original_dir = pathlib.Path("/content/drive/My Drive/DSP566_FinalProjectFolder/Data
new_base_dir = pathlib.Path("/content/drive/My Drive/DSP566_FinalProjectFolder/Data
```

Subsets (Don't rerun)

[ ] ↳ *1 cell hidden*

Setsplits

```
train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(512, 384),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(512, 384),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(512, 384),
    batch_size=32)
```

```
Found 534 files belonging to 6 classes.
Found 114 files belonging to 6 classes.
Found 114 files belonging to 6 classes.
```

```
earlystopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,
                                              restore_best_weights= True,
                                              start_from_epoch = 10)
```

Original Models

Original Model (Funnel In)

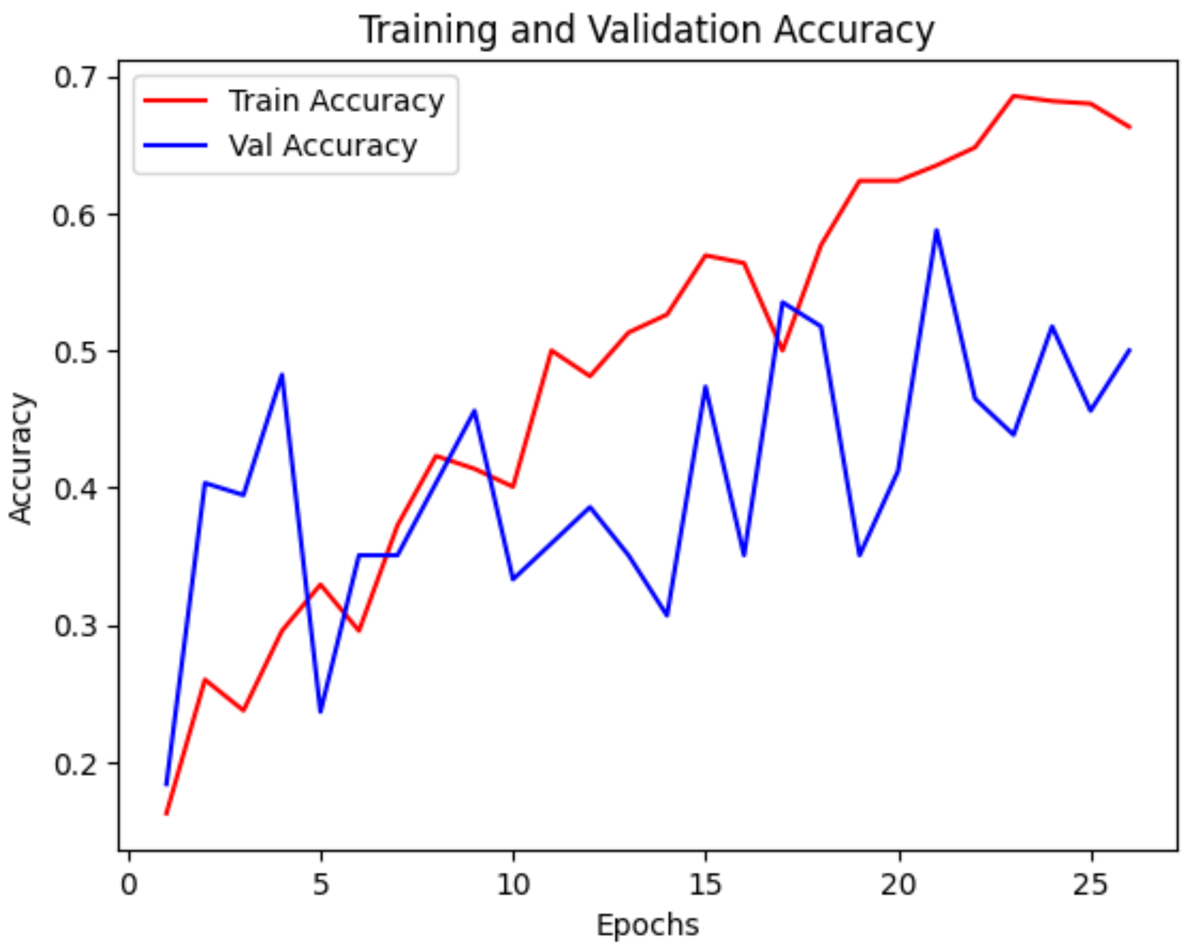
```
inputs = Input(shape=(512, 384, 3))
x = inputs
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(128, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Flatten()(x)
outputs = Dense(6, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)
```

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
history = model.fit(train_dataset, validation_data=validation_dataset, epochs=40,
                    callbacks=[earlystopping])
history_dict = history.history
```

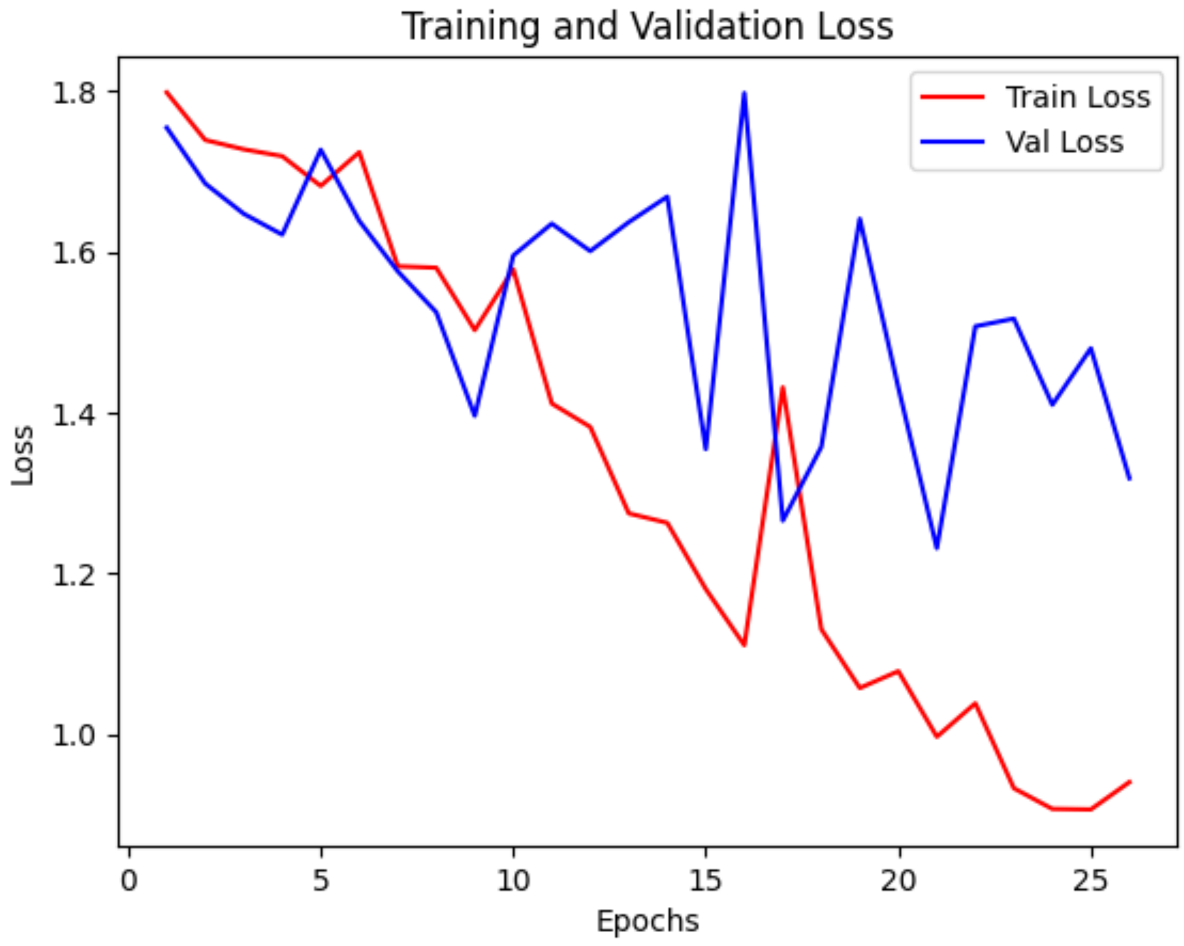
```
Epoch 1/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 187s 10s/step - accuracy: 0.1472 - loss: 1.8132 - v
Epoch 2/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 375ms/step - accuracy: 0.2106 - loss: 1.7509 - v
Epoch 3/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 374ms/step - accuracy: 0.2502 - loss: 1.7235 -
Epoch 4/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 383ms/step - accuracy: 0.2999 - loss: 1.7160 -
Epoch 5/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 7s 420ms/step - accuracy: 0.3433 - loss: 1.6854 - v
Epoch 6/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 379ms/step - accuracy: 0.2842 - loss: 1.7059 -
Epoch 7/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 380ms/step - accuracy: 0.3562 - loss: 1.6136 -
Epoch 8/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 379ms/step - accuracy: 0.4476 - loss: 1.5527 -
Epoch 9/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 7s 421ms/step - accuracy: 0.4079 - loss: 1.4924 - v
Epoch 10/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 372ms/step - accuracy: 0.4740 - loss: 1.4358 - v
Epoch 11/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 7s 382ms/step - accuracy: 0.4760 - loss: 1.4580 - v
Epoch 12/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 422ms/step - accuracy: 0.4820 - loss: 1.3395 -
Epoch 13/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 374ms/step - accuracy: 0.4814 - loss: 1.3224 - v
Epoch 14/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 393ms/step - accuracy: 0.4925 - loss: 1.3353 -
Epoch 15/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 380ms/step - accuracy: 0.5699 - loss: 1.1806 -
Epoch 16/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 7s 422ms/step - accuracy: 0.5836 - loss: 1.0840 - v
Epoch 17/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 380ms/step - accuracy: 0.4216 - loss: 1.6304 -
Epoch 18/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 377ms/step - accuracy: 0.5705 - loss: 1.1348 -
Epoch 19/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 384ms/step - accuracy: 0.6119 - loss: 1.0590 -
Epoch 20/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 391ms/step - accuracy: 0.5737 - loss: 1.1941 -
Epoch 21/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 375ms/step - accuracy: 0.6192 - loss: 0.9996 - v
```

```
Epoch 22/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 385ms/step - accuracy: 0.6289 - loss: 1.0732 -
Epoch 23/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 7s 382ms/step - accuracy: 0.6461 - loss: 0.9528 -
Epoch 24/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 382ms/step - accuracy: 0.6190 - loss: 1.0500 -
Epoch 25/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 376ms/step - accuracy: 0.6670 - loss: 0.9495 -
Epoch 26/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 7s 396ms/step - accuracy: 0.6827 - loss: 0.8980 -
```

```
sequence = range(1, len(history_dict['accuracy']) + 1, 1)
plt.plot(sequence, history_dict['accuracy'], 'r', label='Train Accuracy')
plt.plot(sequence, history_dict['val_accuracy'], 'b', label='Val Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



```
plt.plot(sequence, history_dict['loss'], 'r', label='Train Loss')
plt.plot(sequence, history_dict['val_loss'], 'b', label='Val Loss')
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
test_loss, test_accuracy = model.evaluate(test_dataset)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 28s 7s/step - accuracy: 0.3281 - loss: 1.7913
Test Loss: 1.757429838180542
Test Accuracy: 0.3333333432674408
```

Original Model (Funnel Out)

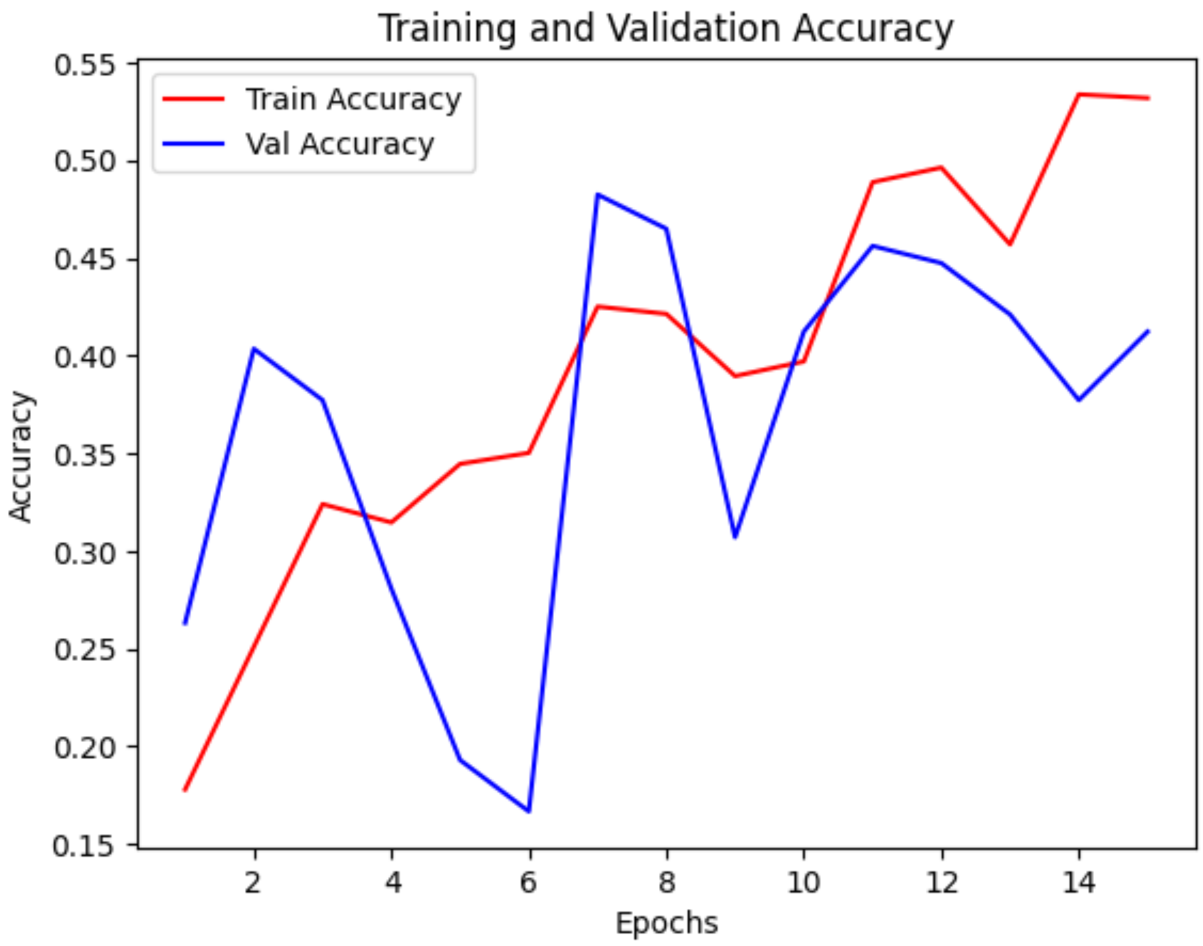
```
inputs = Input(shape=(512, 384, 3))
x = inputs
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Flatten()(x)
outputs = Dense(6, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)

model.summary()
```

```
Model: "functional_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━
┃ Layer (type)                         ┃ Output Shape                ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━
│ input_layer_1 (InputLayer)           │ (None, 512, 384, 3)         │
├──────────────────────────────────────┼─────────────────────────────┼────────
│ rescaling_1 (Rescaling)              │ (None, 512, 384, 3)         │
├──────────────────────────────────────┼─────────────────────────────┼────────
│ conv2d_3 (Conv2D)                    │ (None, 510, 382, 32)        │
├──────────────────────────────────────┼─────────────────────────────┼────────
│ max_pooling2d_3 (MaxPooling2D)       │ (None, 255, 191, 32)        │
├──────────────────────────────────────┼─────────────────────────────┼────────
│ conv2d_4 (Conv2D)                    │ (None, 253, 189, 64)        │
├──────────────────────────────────────┼─────────────────────────────┼────────
│ max_pooling2d_4 (MaxPooling2D)       │ (None, 126, 94, 64)         │
├──────────────────────────────────────┼─────────────────────────────┼────────
│ conv2d_5 (Conv2D)                    │ (None, 124, 92, 128)        │
├──────────────────────────────────────┼─────────────────────────────┼────────
│ max_pooling2d_5 (MaxPooling2D)       │ (None, 62, 46, 128)         │
├──────────────────────────────────────┼─────────────────────────────┼────────
│ flatten_1 (Flatten)                  │ (None, 365056)              │
├──────────────────────────────────────┼─────────────────────────────┼────────
│ dense_1 (Dense)                      │ (None, 6)                   │
└──────────────────────────────────────┴─────────────────────────────┴────────
 Total params: 2,283,590 (8.71 MB)
 Trainable params: 2,283,590 (8.71 MB)
 Non-trainable params: 0 (0.00 B)
```

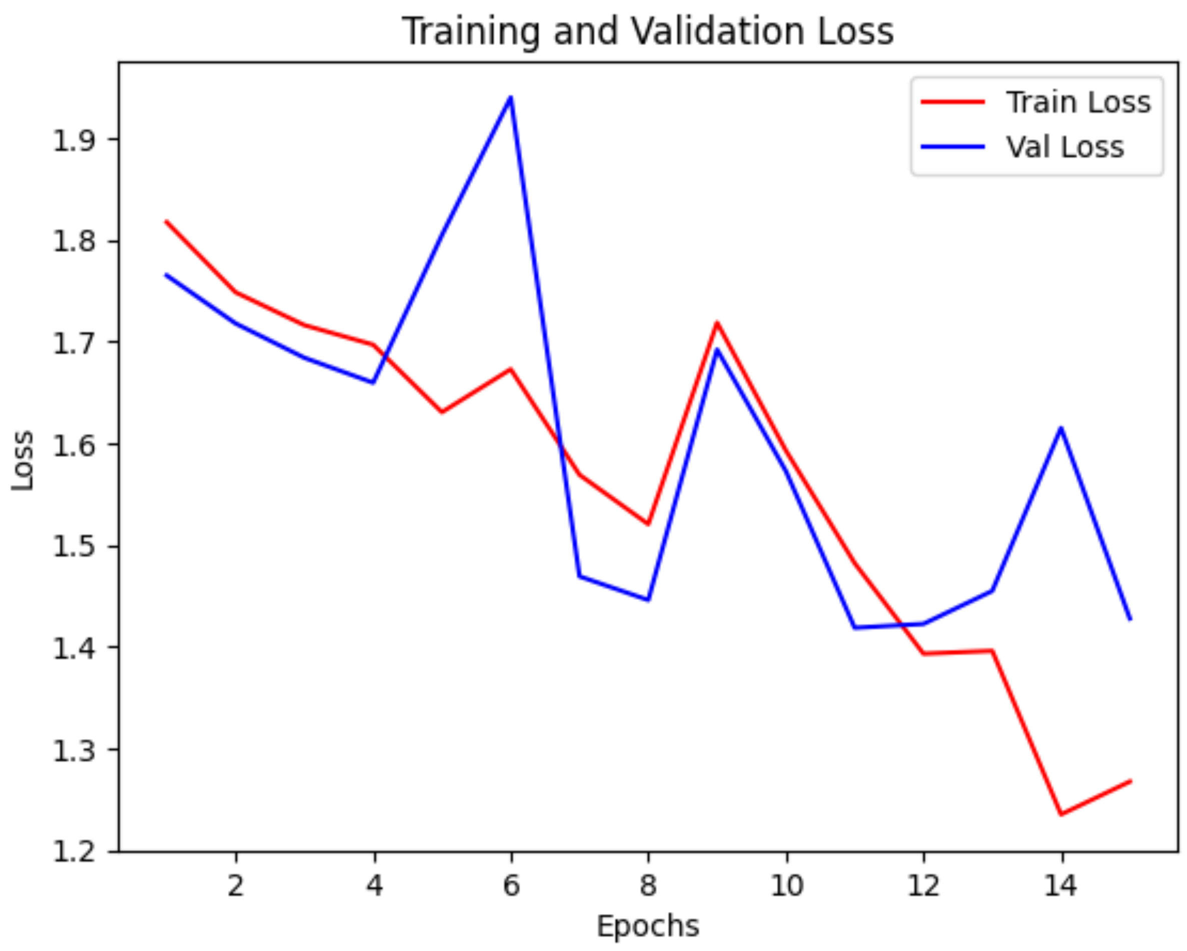
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
history = model.fit(train_dataset, validation_data=validation_dataset, epochs=40,
                    callbacks=[earlystopping])
history_dict = history.history
```

```
Epoch 1/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 31s 1s/step - accuracy: 0.1583 - loss: 1.8556 - va
Epoch 2/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 15s 266ms/step - accuracy: 0.2364 - loss: 1.7535 -
Epoch 3/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 3s 199ms/step - accuracy: 0.3471 - loss: 1.7114 - v
Epoch 4/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 204ms/step - accuracy: 0.3216 - loss: 1.6847 - v
Epoch 5/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 264ms/step - accuracy: 0.3454 - loss: 1.6346 - v
Epoch 6/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 193ms/step - accuracy: 0.3447 - loss: 1.6204 - v
Epoch 7/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 202ms/step - accuracy: 0.3787 - loss: 1.6450 - v
Epoch 8/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 222ms/step - accuracy: 0.4007 - loss: 1.5163 - v
Epoch 9/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 196ms/step - accuracy: 0.4611 - loss: 1.5284 - v
Epoch 10/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 197ms/step - accuracy: 0.3855 - loss: 1.6148 - v
Epoch 11/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 283ms/step - accuracy: 0.4628 - loss: 1.4835 - v
Epoch 12/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 195ms/step - accuracy: 0.5064 - loss: 1.3878 - v
Epoch 13/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 186ms/step - accuracy: 0.4624 - loss: 1.3922 - v
Epoch 14/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 250ms/step - accuracy: 0.5098 - loss: 1.2845 - v
Epoch 15/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 189ms/step - accuracy: 0.4595 - loss: 1.4362 - v
```

```
sequence = range(1, len(history_dict['accuracy']) + 1, 1)
plt.plot(sequence, history_dict['accuracy'], 'r', label='Train Accuracy')
plt.plot(sequence, history_dict['val_accuracy'], 'b', label='Val Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



```
plt.plot(sequence, history_dict['loss'], 'r', label='Train Loss')
plt.plot(sequence, history_dict['val_loss'], 'b', label='Val Loss')
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
test_loss, test_accuracy = model.evaluate(test_dataset)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 114ms/step - accuracy: 0.2218 - loss: 1.7029
Test Loss: 1.6789518594741821
Test Accuracy: 0.2368421107530594
```

Regularization Methods

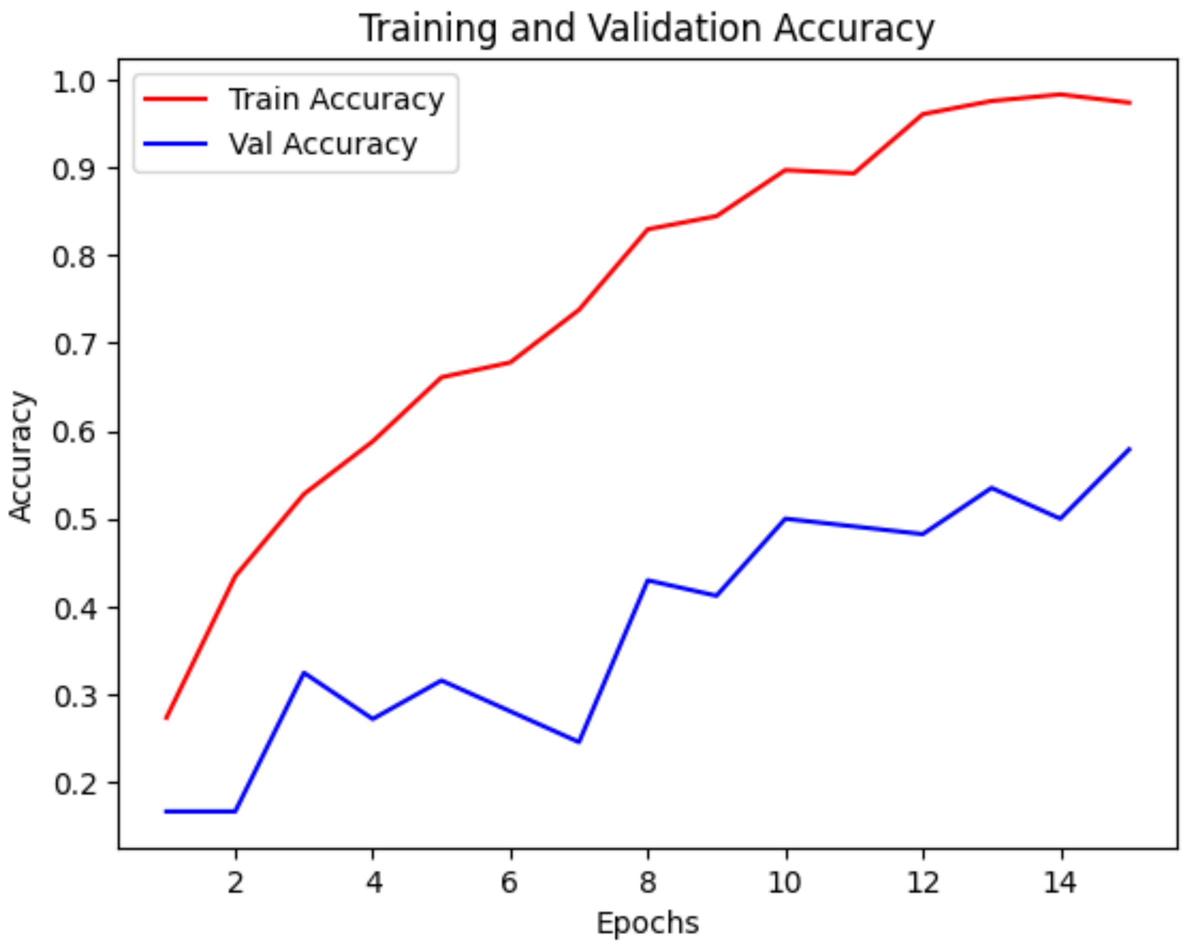
Batch Normalization Test

```
inputs = Input(shape=(512, 384, 3))
x = inputs
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.BatchNormalization()(x)
x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Flatten()(x)
outputs = Dense(6, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)
```

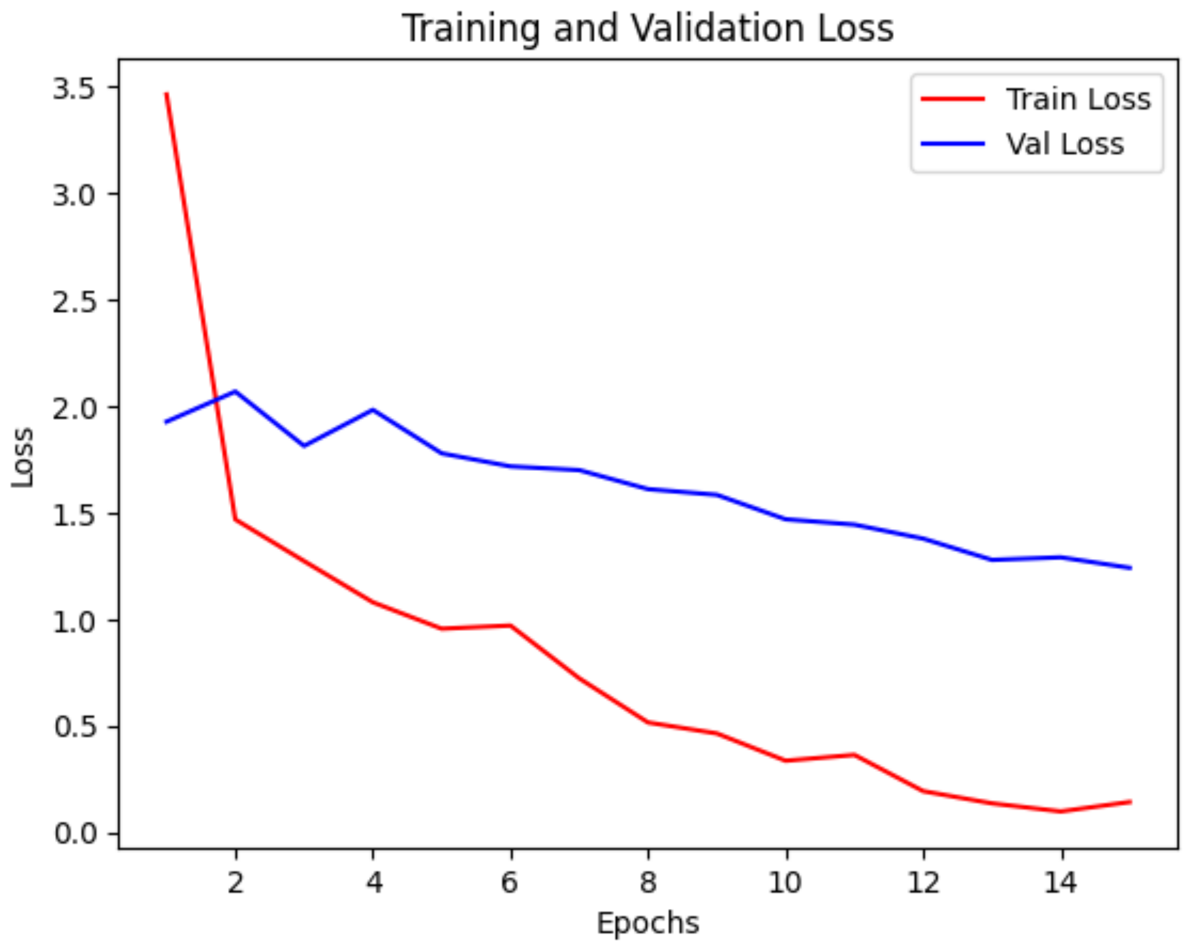
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
history = model.fit(train_dataset, validation_data=validation_dataset, epochs=40,
                    callbacks=[earlystopping])
history_dict = history.history
```

```
Epoch 1/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 455ms/step - accuracy: 0.2256 - loss: 5.1848 -
Epoch 2/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 3s 191ms/step - accuracy: 0.3955 - loss: 1.5228 -
Epoch 3/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 3s 197ms/step - accuracy: 0.5324 - loss: 1.2539 -
Epoch 4/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 303ms/step - accuracy: 0.5755 - loss: 1.1297 -
Epoch 5/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 206ms/step - accuracy: 0.6378 - loss: 1.0324 -
Epoch 6/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 203ms/step - accuracy: 0.6411 - loss: 1.0959 -
Epoch 7/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 226ms/step - accuracy: 0.7258 - loss: 0.7988 -
Epoch 8/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 205ms/step - accuracy: 0.8192 - loss: 0.5406 -
Epoch 9/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 213ms/step - accuracy: 0.8539 - loss: 0.4652 -
Epoch 10/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 199ms/step - accuracy: 0.9061 - loss: 0.3483 -
Epoch 11/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 198ms/step - accuracy: 0.9185 - loss: 0.3158 -
Epoch 12/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 277ms/step - accuracy: 0.9559 - loss: 0.2229 -
Epoch 13/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 206ms/step - accuracy: 0.9701 - loss: 0.1542 -
Epoch 14/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 206ms/step - accuracy: 0.9832 - loss: 0.1036 -
Epoch 15/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 239ms/step - accuracy: 0.9668 - loss: 0.1652 -
```

```
sequence = range(1, len(history_dict['accuracy']) + 1, 1)
plt.plot(sequence, history_dict['accuracy'], 'r', label='Train Accuracy')
plt.plot(sequence, history_dict['val_accuracy'], 'b', label='Val Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



```
plt.plot(sequence, history_dict['loss'], 'r', label='Train Loss')
plt.plot(sequence, history_dict['val_loss'], 'b', label='Val Loss')
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
test_loss, test_accuracy = model.evaluate(test_dataset)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - accuracy: 0.3376 - loss: 1.6401
Test Loss: 1.6309483051300049
Test Accuracy: 0.359649121761322
```

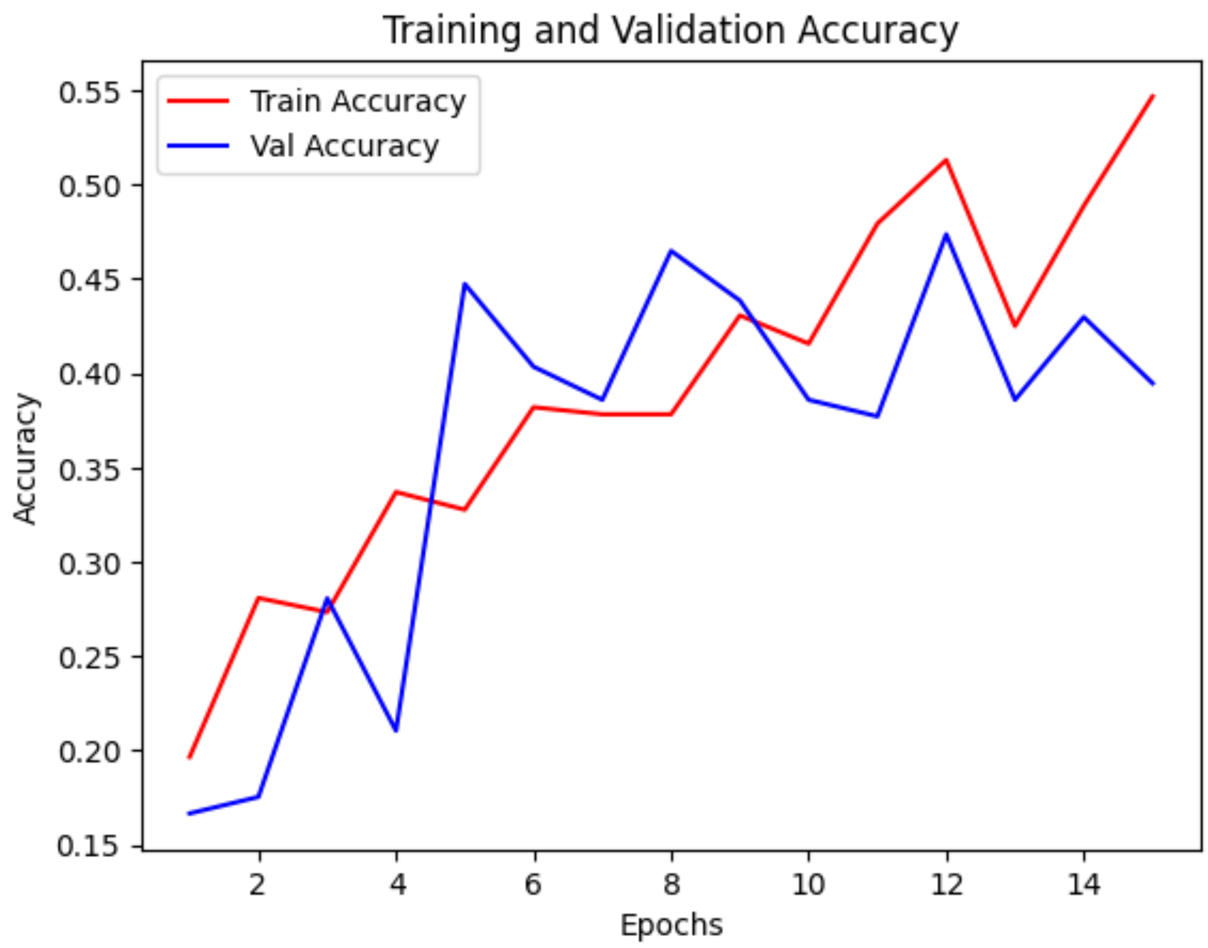
L2 Regularizer Test

```
inputs = Input(shape=(512, 384, 3))
x = inputs
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dense(64, activation= 'relu', kernel_regularizer= keras.regularizers.l
x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Flatten()(x)
outputs = Dense(6, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)
```

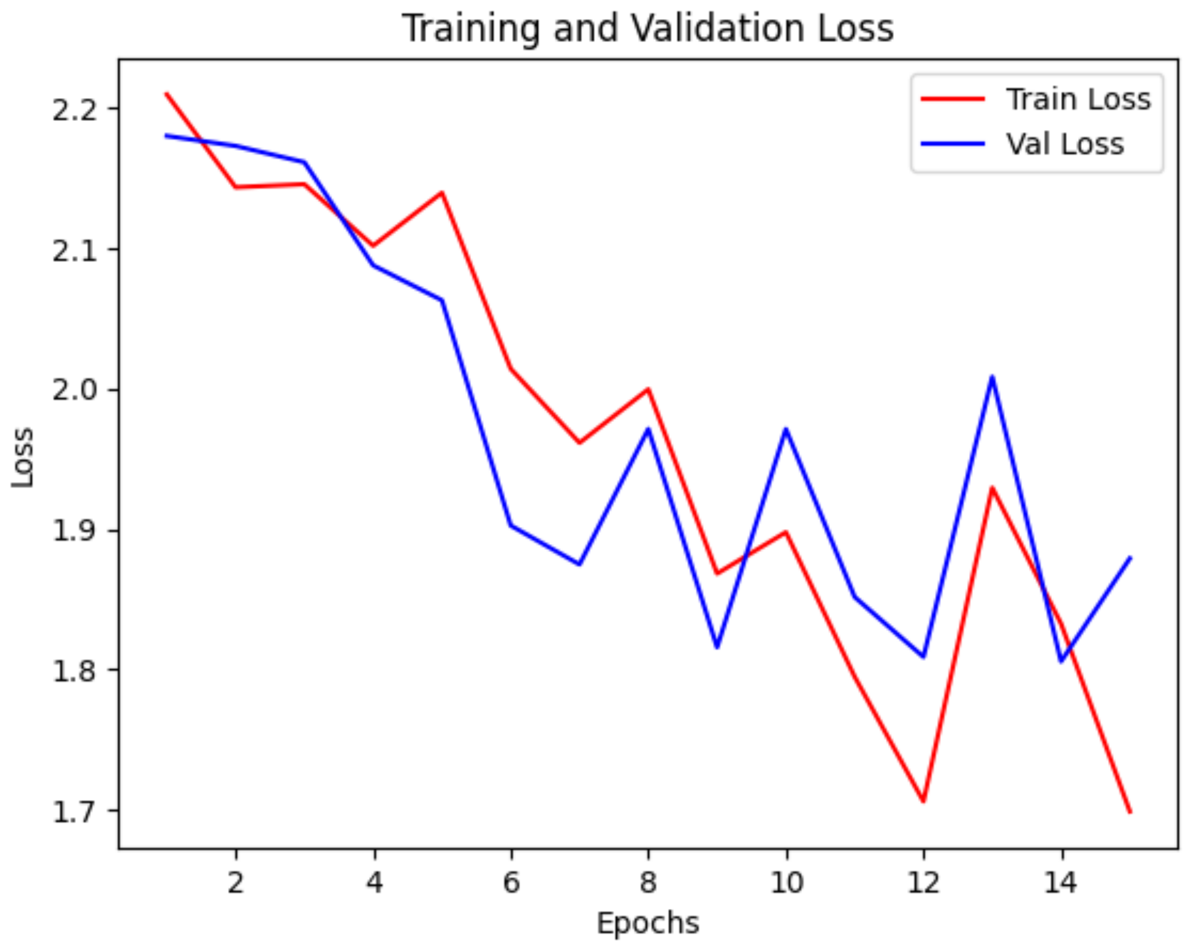
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
history = model.fit(train_dataset, validation_data=validation_dataset, epochs=40,
                    callbacks=[earlystopping])
history_dict = history.history
```

```
Epoch 1/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 28s 1s/step - accuracy: 0.1739 - loss: 2.2168 - va
Epoch 2/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 18s 245ms/step - accuracy: 0.2138 - loss: 2.1664 -
Epoch 3/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 275ms/step - accuracy: 0.3025 - loss: 2.0952 - v
Epoch 4/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 247ms/step - accuracy: 0.3538 - loss: 2.1128 - v
Epoch 5/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 246ms/step - accuracy: 0.3247 - loss: 2.1107 - v
Epoch 6/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 276ms/step - accuracy: 0.3878 - loss: 2.0411 - v
Epoch 7/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 246ms/step - accuracy: 0.3591 - loss: 1.9817 - v
Epoch 8/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 247ms/step - accuracy: 0.4311 - loss: 1.9055 - v
Epoch 9/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 287ms/step - accuracy: 0.4317 - loss: 1.8954 - v
Epoch 10/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 253ms/step - accuracy: 0.4440 - loss: 1.7967 - v
Epoch 11/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 256ms/step - accuracy: 0.4821 - loss: 1.8042 - v
Epoch 12/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 276ms/step - accuracy: 0.4988 - loss: 1.7521 - v
Epoch 13/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 246ms/step - accuracy: 0.4638 - loss: 1.7486 - v
Epoch 14/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 248ms/step - accuracy: 0.4590 - loss: 1.8783 - v
Epoch 15/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 270ms/step - accuracy: 0.5470 - loss: 1.6839 - v
```

```
sequence = range(1, len(history_dict['accuracy']) + 1, 1)
plt.plot(sequence, history_dict['accuracy'], 'r', label='Train Accuracy')
plt.plot(sequence, history_dict['val_accuracy'], 'b', label='Val Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



```
plt.plot(sequence, history_dict['loss'], 'r', label='Train Loss')
plt.plot(sequence, history_dict['val_loss'], 'b', label='Val Loss')
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
test_loss, test_accuracy = model.evaluate(test_dataset)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

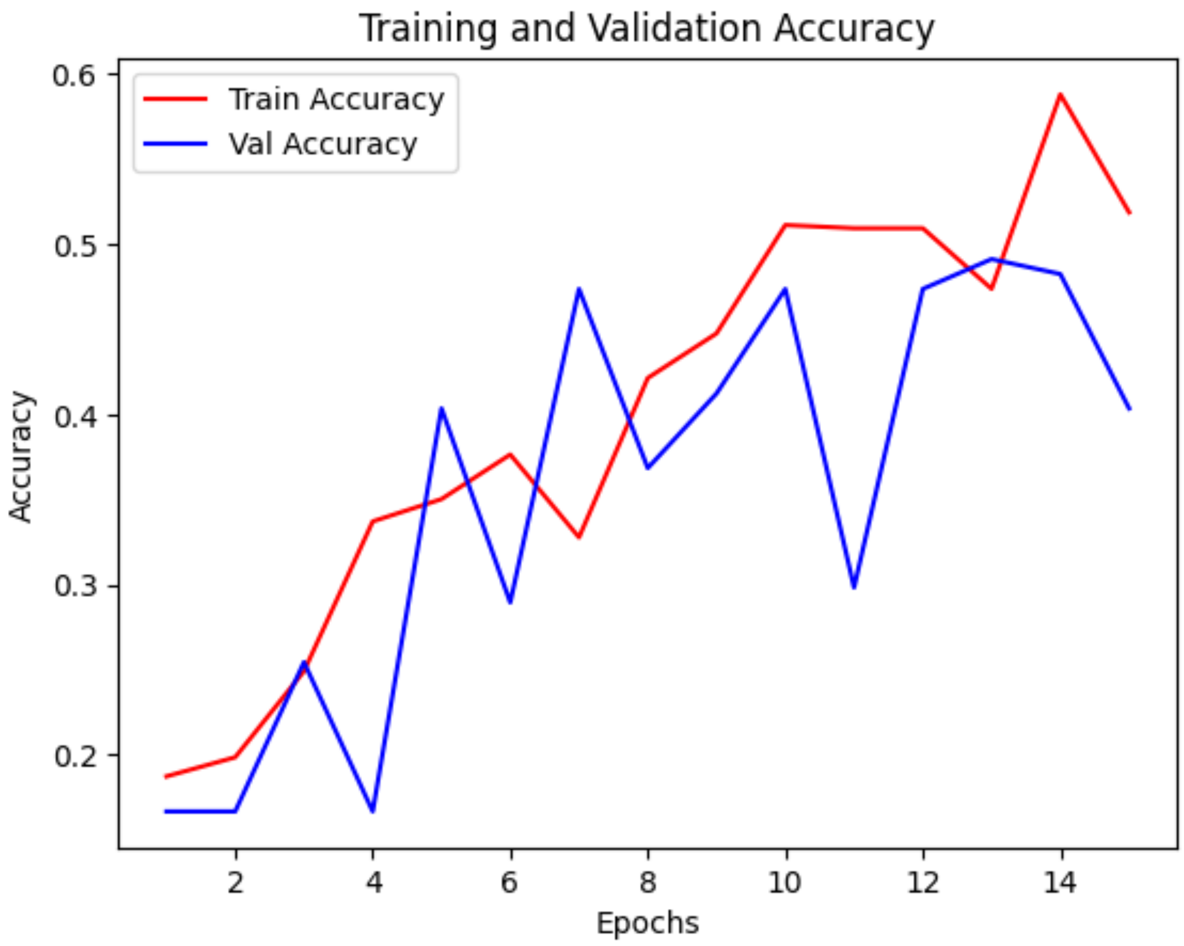
```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 125ms/step - accuracy: 0.2643 - loss: 2.2046
Test Loss: 2.2129781246185303
Test Accuracy: 0.25438597798347473
```

L1 Regularizer Test

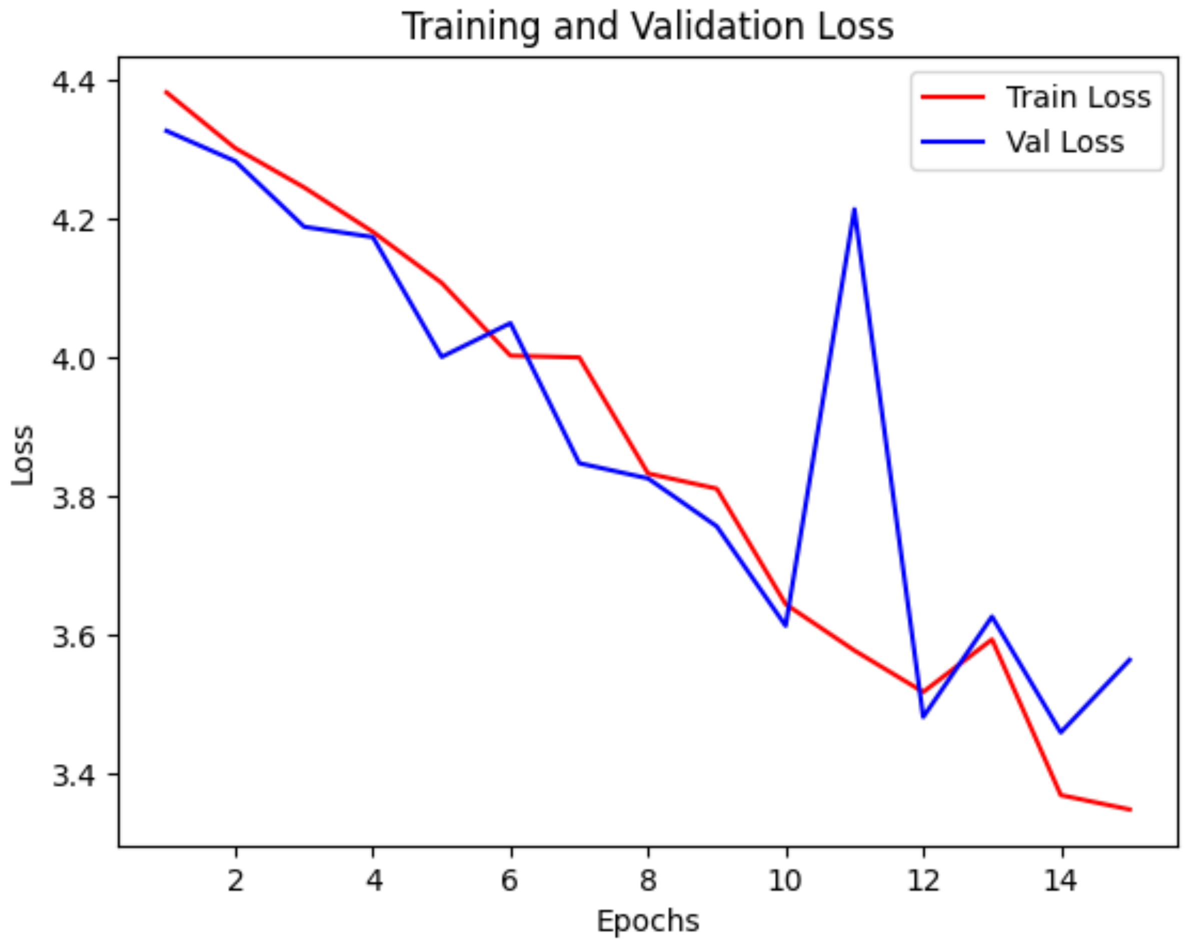
```
inputs = Input(shape=(512, 384, 3))
x = inputs
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dense(64, activation= 'relu', kernel_regularizer= keras.regularizers.l
x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Flatten()(x)
outputs = Dense(6, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)
```

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
history = model.fit(train_dataset, validation_data=validation_dataset, epochs=40,
                    callbacks=[earlystopping])
history_dict = history.history
```

```
Epoch 1/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 441ms/step - accuracy: 0.1768 - loss: 4.4075 - \
Epoch 2/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 255ms/step - accuracy: 0.2113 - loss: 4.3060 - \
Epoch 3/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 246ms/step - accuracy: 0.2376 - loss: 4.2595 - \
Epoch 4/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 277ms/step - accuracy: 0.3563 - loss: 4.1820 - \
Epoch 5/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 255ms/step - accuracy: 0.3311 - loss: 4.1010 - \
Epoch 6/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 255ms/step - accuracy: 0.3588 - loss: 4.0183 - \
Epoch 7/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 312ms/step - accuracy: 0.2856 - loss: 4.0359 - \
Epoch 8/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 246ms/step - accuracy: 0.4388 - loss: 3.8452 - \
Epoch 9/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 297ms/step - accuracy: 0.4133 - loss: 3.8246 - \
Epoch 10/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 267ms/step - accuracy: 0.4979 - loss: 3.6638 -
Epoch 11/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 263ms/step - accuracy: 0.5099 - loss: 3.6073 - \
Epoch 12/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 247ms/step - accuracy: 0.4602 - loss: 3.5996 - \
Epoch 13/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 270ms/step - accuracy: 0.5178 - loss: 3.4797 - \
Epoch 14/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 247ms/step - accuracy: 0.5678 - loss: 3.4317 - \
Epoch 15/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 254ms/step - accuracy: 0.5187 - loss: 3.3160 - \
sequence = range(1, len(history_dict['accuracy']) + 1, 1)
plt.plot(sequence, history_dict['accuracy'], 'r', label='Train Accuracy')
plt.plot(sequence, history_dict['val_accuracy'], 'b', label='Val Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



```
plt.plot(sequence, history_dict['loss'], 'r', label='Train Loss')
plt.plot(sequence, history_dict['val_loss'], 'b', label='Val Loss')
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
test_loss, test_accuracy = model.evaluate(test_dataset)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 180ms/step - accuracy: 0.1712 - loss: 4.5829
Test Loss: 4.610139846801758
Test Accuracy: 0.17543859779834747
```

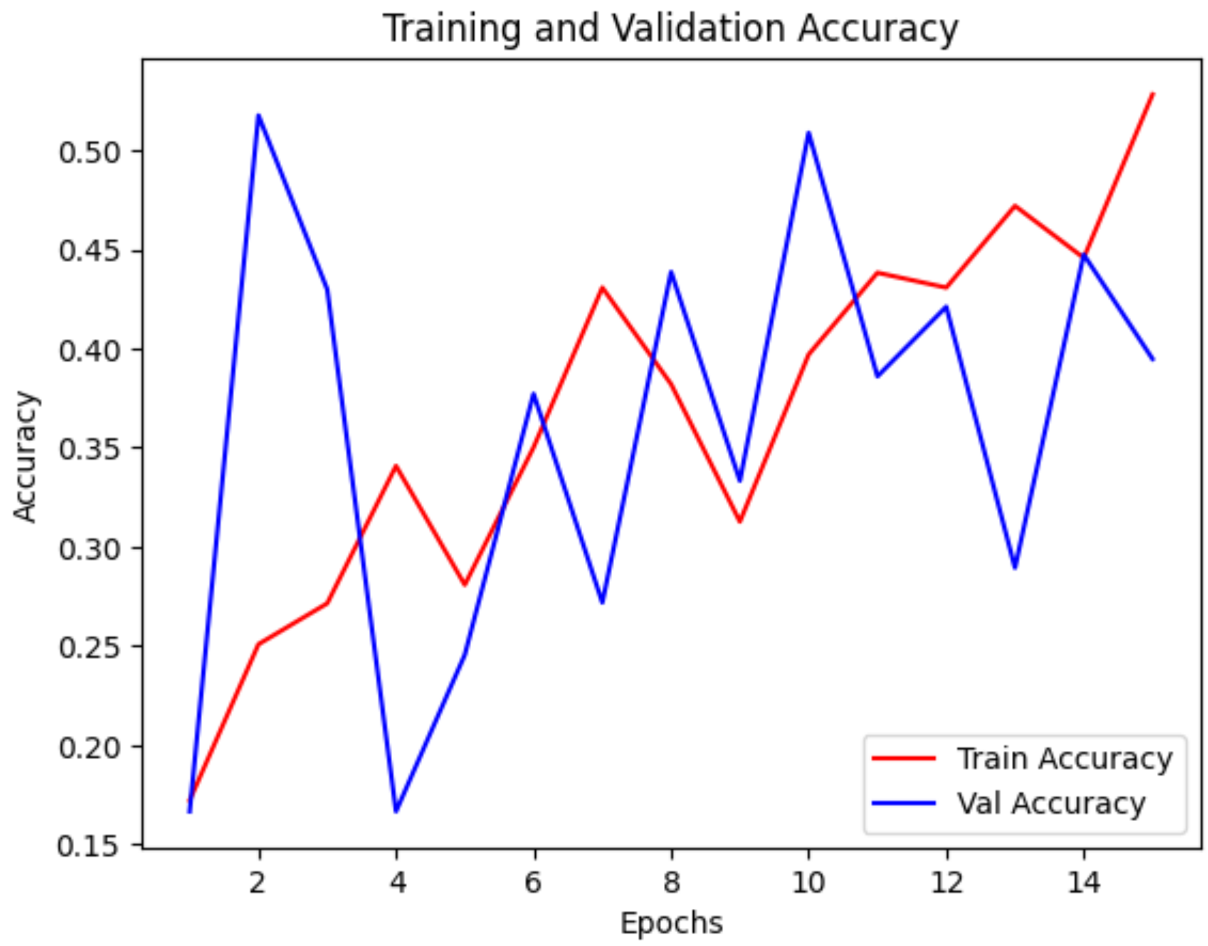
L1 and L2 Regularizer

```
inputs = Input(shape=(512, 384, 3))
x = inputs
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dense(64, activation= 'relu', kernel_regularizer= keras.regularizers.L
x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Flatten()(x)
outputs = Dense(6, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)
```

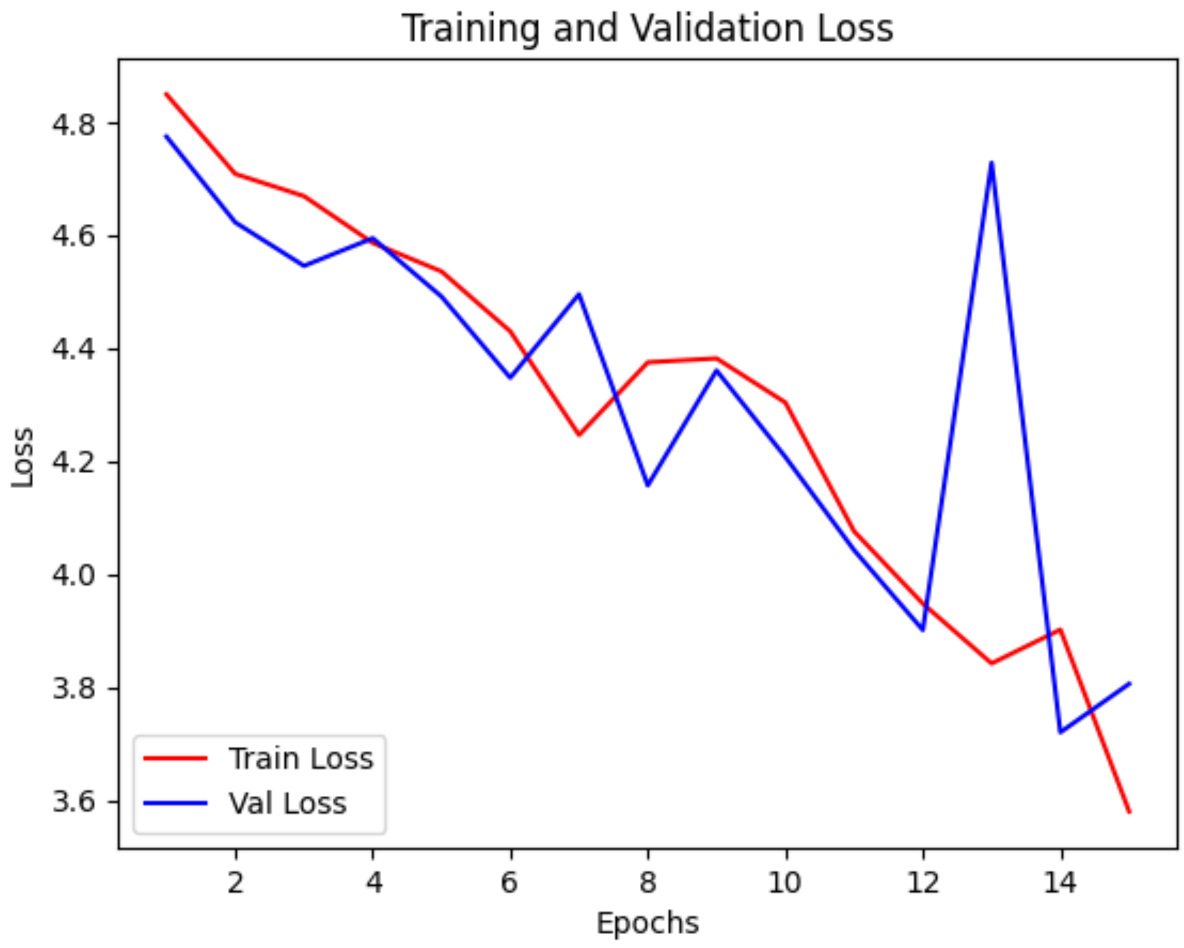
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
history = model.fit(train_dataset, validation_data=validation_dataset, epochs=40,
                    callbacks=[earlystopping])
history_dict = history.history
```

```
Epoch 1/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 443ms/step - accuracy: 0.1595 - loss: 4.8864 -
Epoch 2/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 275ms/step - accuracy: 0.2107 - loss: 4.7434 - 
Epoch 3/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 251ms/step - accuracy: 0.2847 - loss: 4.6751 - 
Epoch 4/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 251ms/step - accuracy: 0.3610 - loss: 4.5174 - 
Epoch 5/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 298ms/step - accuracy: 0.2378 - loss: 4.5534 - 
Epoch 6/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 250ms/step - accuracy: 0.3261 - loss: 4.4336 - 
Epoch 7/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 245ms/step - accuracy: 0.4261 - loss: 4.2917 - 
Epoch 8/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 315ms/step - accuracy: 0.2950 - loss: 4.5735 - 
Epoch 9/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 250ms/step - accuracy: 0.3950 - loss: 4.2697 - 
Epoch 10/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 254ms/step - accuracy: 0.3526 - loss: 4.3329 - 
Epoch 11/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 260ms/step - accuracy: 0.4250 - loss: 4.1368 - 
Epoch 12/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 252ms/step - accuracy: 0.4128 - loss: 3.9916 - 
Epoch 13/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 261ms/step - accuracy: 0.4738 - loss: 3.8403 - 
Epoch 14/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 260ms/step - accuracy: 0.3954 - loss: 4.0798 - 
Epoch 15/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 252ms/step - accuracy: 0.5478 - loss: 3.5936 - 
```

```
sequence = range(1, len(history_dict['accuracy']) + 1, 1)
plt.plot(sequence, history_dict['accuracy'], 'r', label='Train Accuracy')
plt.plot(sequence, history_dict['val_accuracy'], 'b', label='Val Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



```
plt.plot(sequence, history_dict['loss'], 'r', label='Train Loss')
plt.plot(sequence, history_dict['val_loss'], 'b', label='Val Loss')
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
test_loss, test_accuracy = model.evaluate(test_dataset)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 110ms/step - accuracy: 0.3383 - loss: 4.2472
Test Loss: 4.266404628753662
Test Accuracy: 0.35087719559669495
```

Dropout

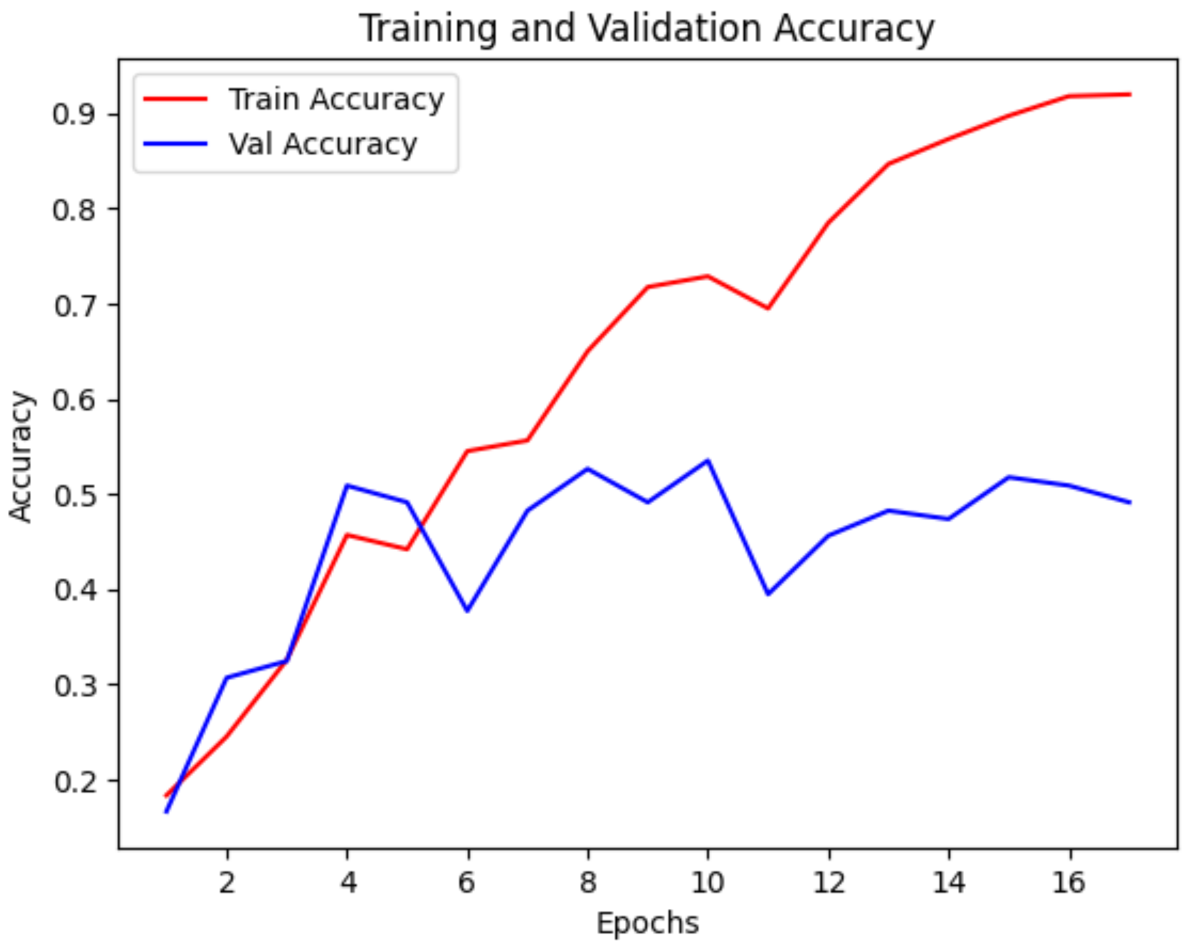
```
Start coding or generate with AI.
```

```
inputs = Input(shape=(512, 384, 3))
x = inputs
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dropout(0.4)(x)
x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dropout(0.4)(x)
x = layers.Conv2D(128, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Flatten()(x)
outputs = Dense(6, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)
```

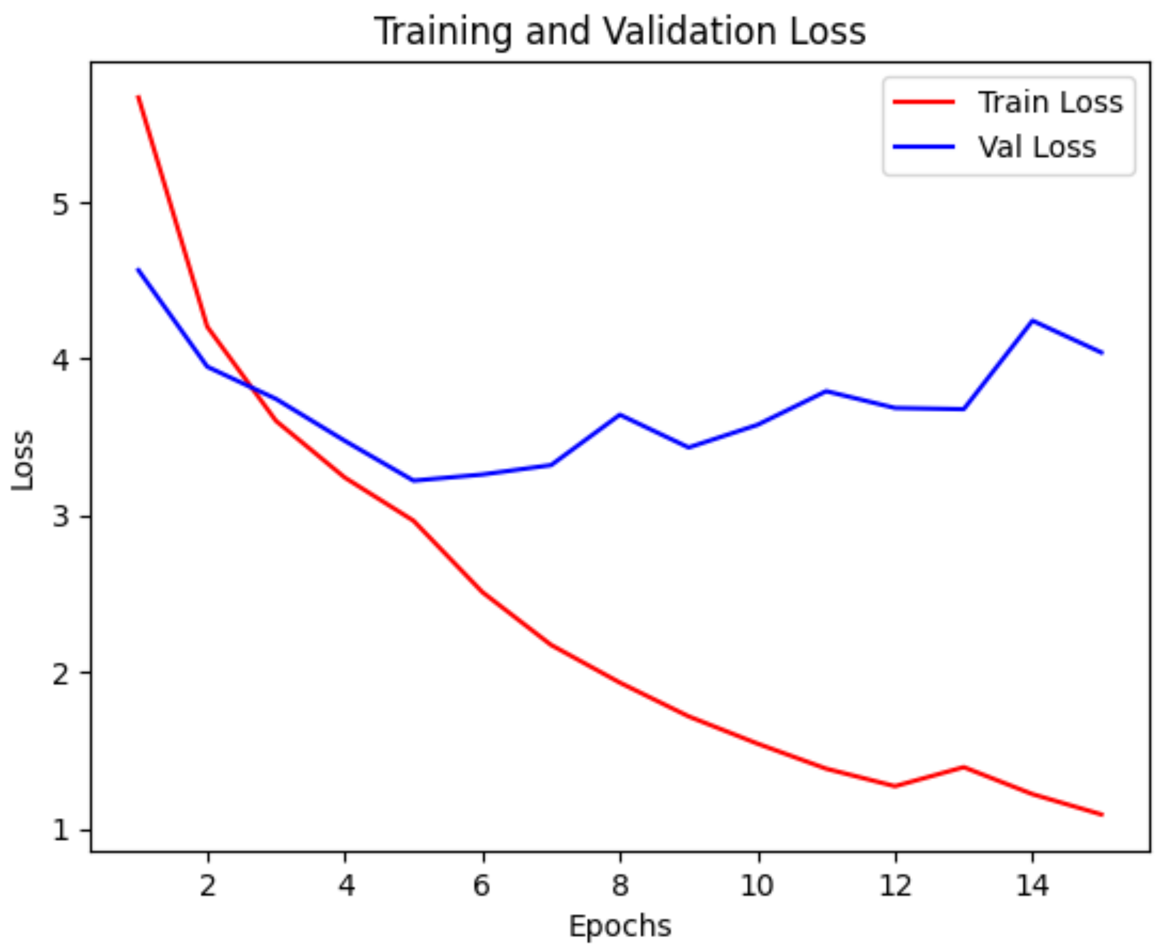
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
history = model.fit(train_dataset, validation_data=validation_dataset, epochs=40,
                    callbacks=[earlystopping])
history_dict = history.history
```

```
Epoch 1/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step - accuracy: 0.1542 - loss: 21.8816 - va
Epoch 2/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 21s 198ms/step - accuracy: 0.2001 - loss: 1.7847 -
Epoch 3/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 245ms/step - accuracy: 0.1597 - loss: 1.7527 - v
Epoch 4/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 203ms/step - accuracy: 0.2174 - loss: 1.6460 - v
Epoch 5/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 250ms/step - accuracy: 0.4271 - loss: 1.5073 - v
Epoch 6/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 203ms/step - accuracy: 0.4873 - loss: 1.3894 - v
Epoch 7/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 3s 200ms/step - accuracy: 0.5590 - loss: 1.2239 - v
Epoch 8/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 240ms/step - accuracy: 0.6195 - loss: 1.1025 - v
Epoch 9/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 207ms/step - accuracy: 0.6185 - loss: 1.0929 - v
Epoch 10/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 207ms/step - accuracy: 0.6857 - loss: 0.8808 - v
Epoch 11/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 286ms/step - accuracy: 0.7021 - loss: 0.7951 - v
Epoch 12/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 212ms/step - accuracy: 0.7362 - loss: 0.7485 - v
Epoch 13/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 199ms/step - accuracy: 0.7631 - loss: 0.7338 - v
Epoch 14/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 265ms/step - accuracy: 0.7871 - loss: 0.5736 - v
Epoch 15/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 200ms/step - accuracy: 0.8268 - loss: 0.6325 - v
Epoch 16/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 248ms/step - accuracy: 0.7742 - loss: 0.8088 - v
```

```
sequence = range(1, len(history_dict['accuracy']) + 1, 1)
plt.plot(sequence, history_dict['accuracy'], 'r', label='Train Accuracy')
plt.plot(sequence, history_dict['val_accuracy'], 'b', label='Val Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



```
plt.plot(sequence, history_dict['loss'], 'r', label='Train Loss')
plt.plot(sequence, history_dict['val_loss'], 'b', label='Val Loss')
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
test_loss, test_accuracy = model.evaluate(test_dataset)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 125ms/step - accuracy: 0.2849 - loss: 5.1118
Test Loss: 4.998693943023682
Test Accuracy: 0.2982456088066101
```

Optimizers

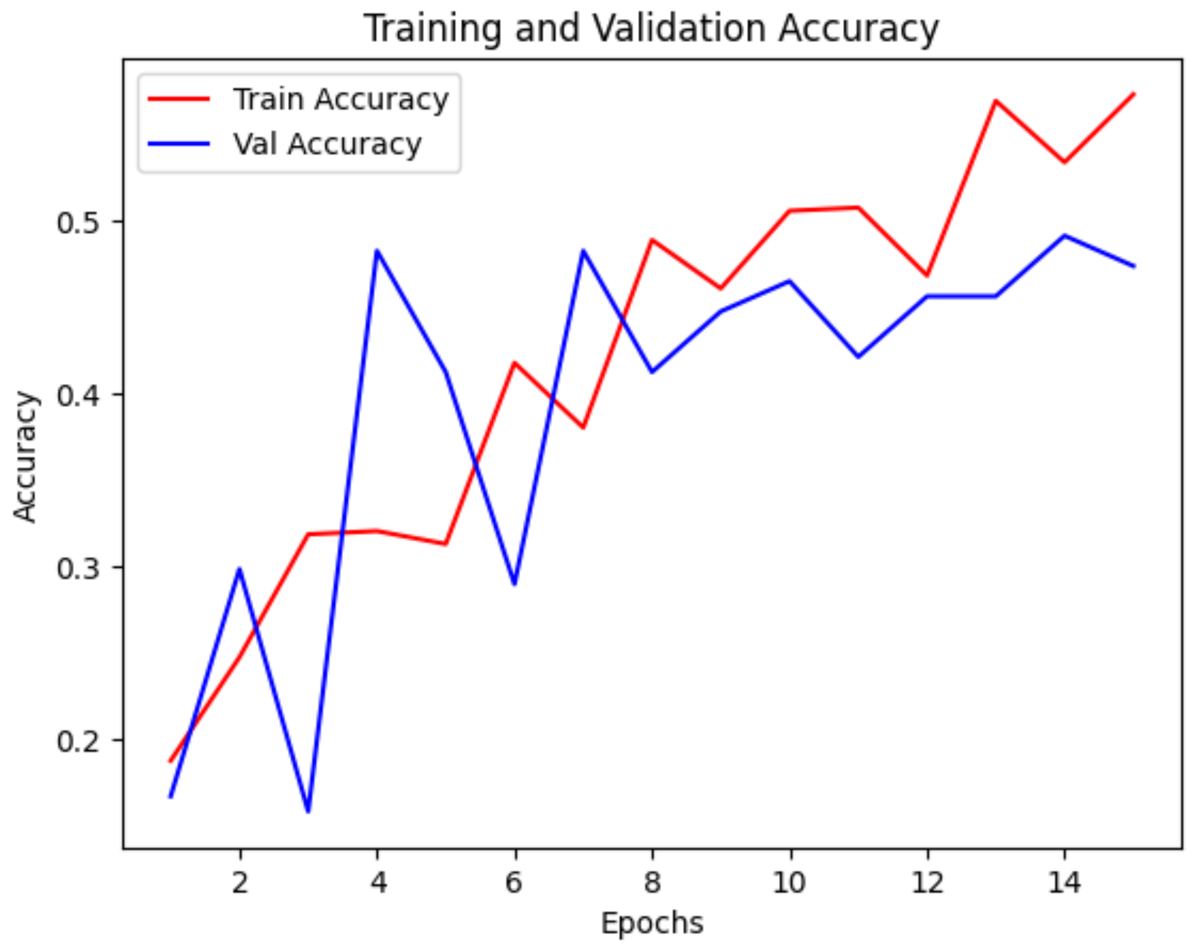
SGD

```
inputs = Input(shape=(512, 384, 3))
x = inputs
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dense(64, activation= 'relu', kernel_regularizer= keras.regularizers.L
x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Flatten()(x)
outputs = Dense(6, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)
```

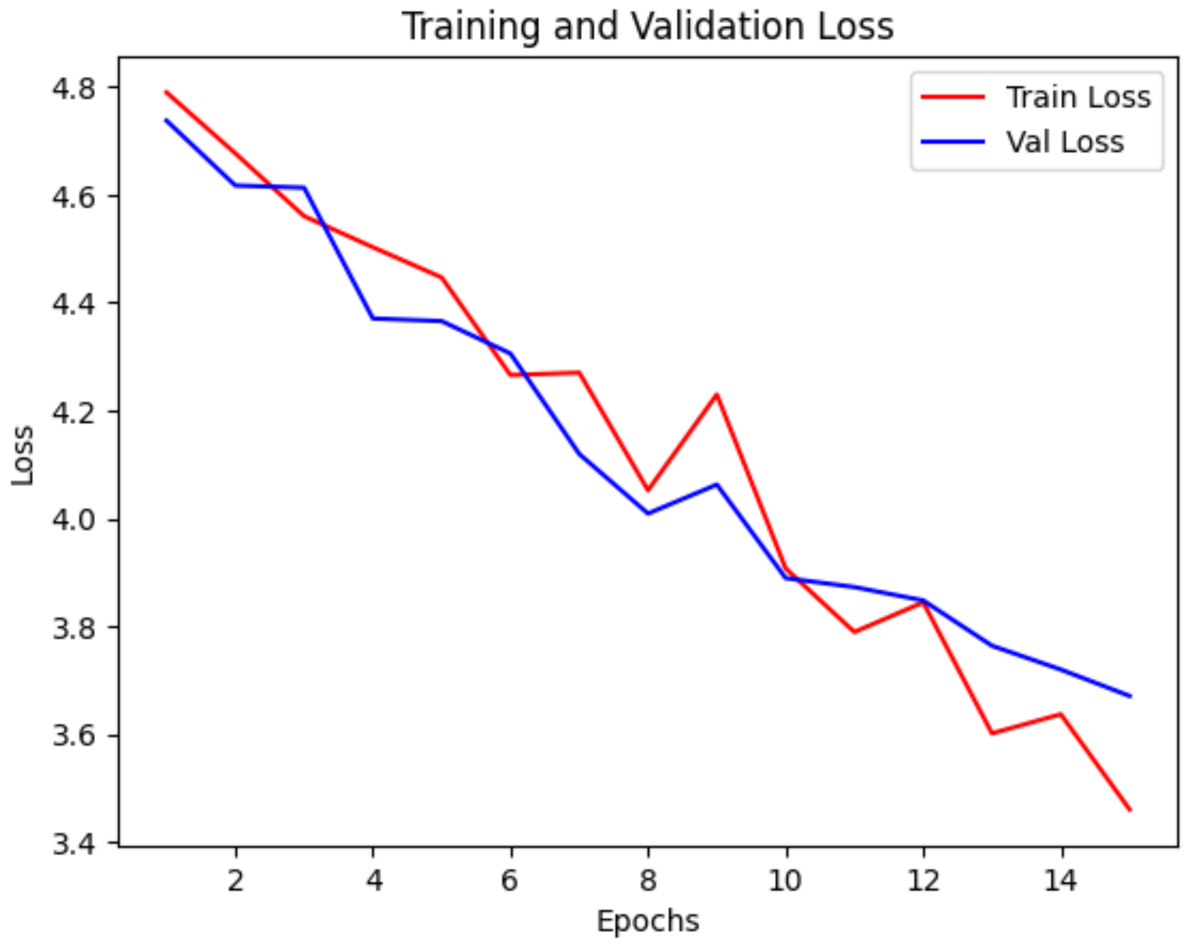
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
history = model.fit(train_dataset, validation_data=validation_dataset, epochs=40,
                    callbacks=[earlystopping])
history_dict = history.history
```

```
Epoch 1/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 438ms/step - accuracy: 0.1711 - loss: 4.8112 - 
Epoch 2/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 246ms/step - accuracy: 0.2473 - loss: 4.6951 - 
Epoch 3/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 301ms/step - accuracy: 0.3329 - loss: 4.5692 - 
Epoch 4/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 251ms/step - accuracy: 0.3037 - loss: 4.5275 - 
Epoch 5/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 283ms/step - accuracy: 0.3402 - loss: 4.4487 - 
Epoch 6/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 251ms/step - accuracy: 0.4025 - loss: 4.3104 - 
Epoch 7/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 250ms/step - accuracy: 0.3636 - loss: 4.3239 - 
Epoch 8/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 286ms/step - accuracy: 0.4712 - loss: 4.1170 - 
Epoch 9/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 243ms/step - accuracy: 0.4346 - loss: 4.2641 - 
Epoch 10/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 244ms/step - accuracy: 0.5067 - loss: 3.9676 - 
Epoch 11/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 284ms/step - accuracy: 0.4978 - loss: 3.7646 - 
Epoch 12/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 252ms/step - accuracy: 0.5031 - loss: 3.8099 - 
Epoch 13/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 251ms/step - accuracy: 0.5545 - loss: 3.6716 - 
Epoch 14/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 291ms/step - accuracy: 0.5294 - loss: 3.6246 - 
Epoch 15/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 245ms/step - accuracy: 0.5714 - loss: 3.4743 - 
```

```
sequence = range(1, len(history_dict['accuracy']) + 1, 1)
plt.plot(sequence, history_dict['accuracy'], 'r', label='Train Accuracy')
plt.plot(sequence, history_dict['val_accuracy'], 'b', label='Val Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



```
plt.plot(sequence, history_dict['loss'], 'r', label='Train Loss')
plt.plot(sequence, history_dict['val_loss'], 'b', label='Val Loss')
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
test_loss, test_accuracy = model.evaluate(test_dataset)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 116ms/step - accuracy: 0.2847 - loss: 4.2170
Test Loss: 4.258544445037842
Test Accuracy: 0.24561403691768646
```

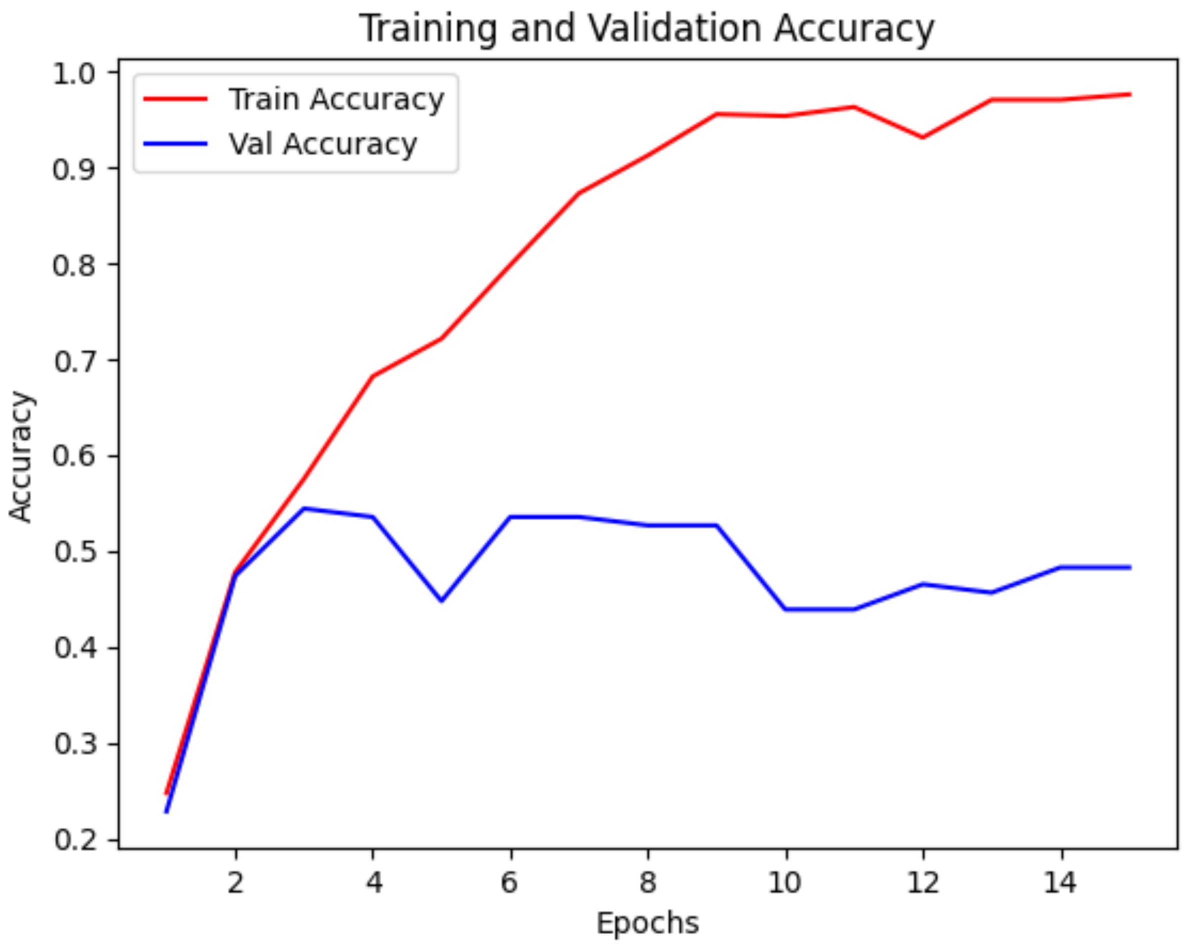
Adam

```
inputs = Input(shape=(512, 384, 3))
x = inputs
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dense(64, activation= 'relu', kernel_regularizer= keras.regularizers.L
x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Flatten()(x)
outputs = Dense(6, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)
```

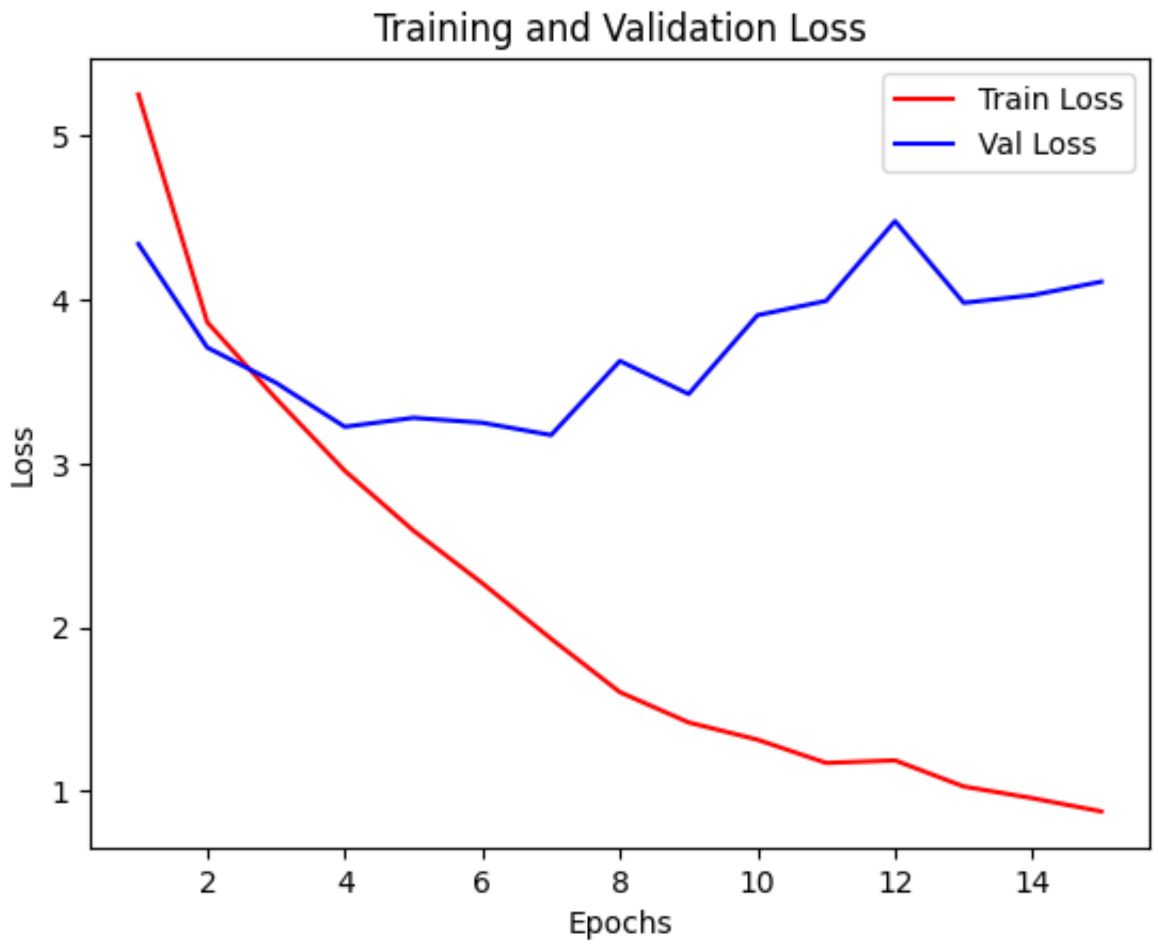
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
history = model.fit(train_dataset, validation_data=validation_dataset, epochs=40,
                    callbacks=[earlystopping])
history_dict = history.history
```

```
Epoch 1/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 458ms/step - accuracy: 0.1795 - loss: 6.2203 -
Epoch 2/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 254ms/step - accuracy: 0.4252 - loss: 4.0128 -
Epoch 3/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 280ms/step - accuracy: 0.5810 - loss: 3.4271 -
Epoch 4/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 254ms/step - accuracy: 0.6497 - loss: 3.0733 -
Epoch 5/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 266ms/step - accuracy: 0.7268 - loss: 2.6148 -
Epoch 6/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 281ms/step - accuracy: 0.7651 - loss: 2.3734 -
Epoch 7/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 247ms/step - accuracy: 0.8599 - loss: 1.9878 -
Epoch 8/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 268ms/step - accuracy: 0.8824 - loss: 1.6749 -
Epoch 9/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 256ms/step - accuracy: 0.9530 - loss: 1.4555 -
Epoch 10/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 256ms/step - accuracy: 0.9569 - loss: 1.3186 -
Epoch 11/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 303ms/step - accuracy: 0.9720 - loss: 1.1757 -
Epoch 12/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 254ms/step - accuracy: 0.9167 - loss: 1.2319 -
Epoch 13/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 288ms/step - accuracy: 0.9710 - loss: 1.0174 -
Epoch 14/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 255ms/step - accuracy: 0.9685 - loss: 0.9753 -
Epoch 15/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 255ms/step - accuracy: 0.9759 - loss: 0.8824 -
```

```
sequence = range(1, len(history_dict['accuracy']) + 1, 1)
plt.plot(sequence, history_dict['accuracy'], 'r', label='Train Accuracy')
plt.plot(sequence, history_dict['val_accuracy'], 'b', label='Val Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



```
plt.plot(sequence, history_dict['loss'], 'r', label='Train Loss')
plt.plot(sequence, history_dict['val_loss'], 'b', label='Val Loss')
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
test_loss, test_accuracy = model.evaluate(test_dataset)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 207ms/step - accuracy: 0.3560 - loss: 4.7130
Test Loss: 4.903677940368652
Test Accuracy: 0.35087719559669495
```

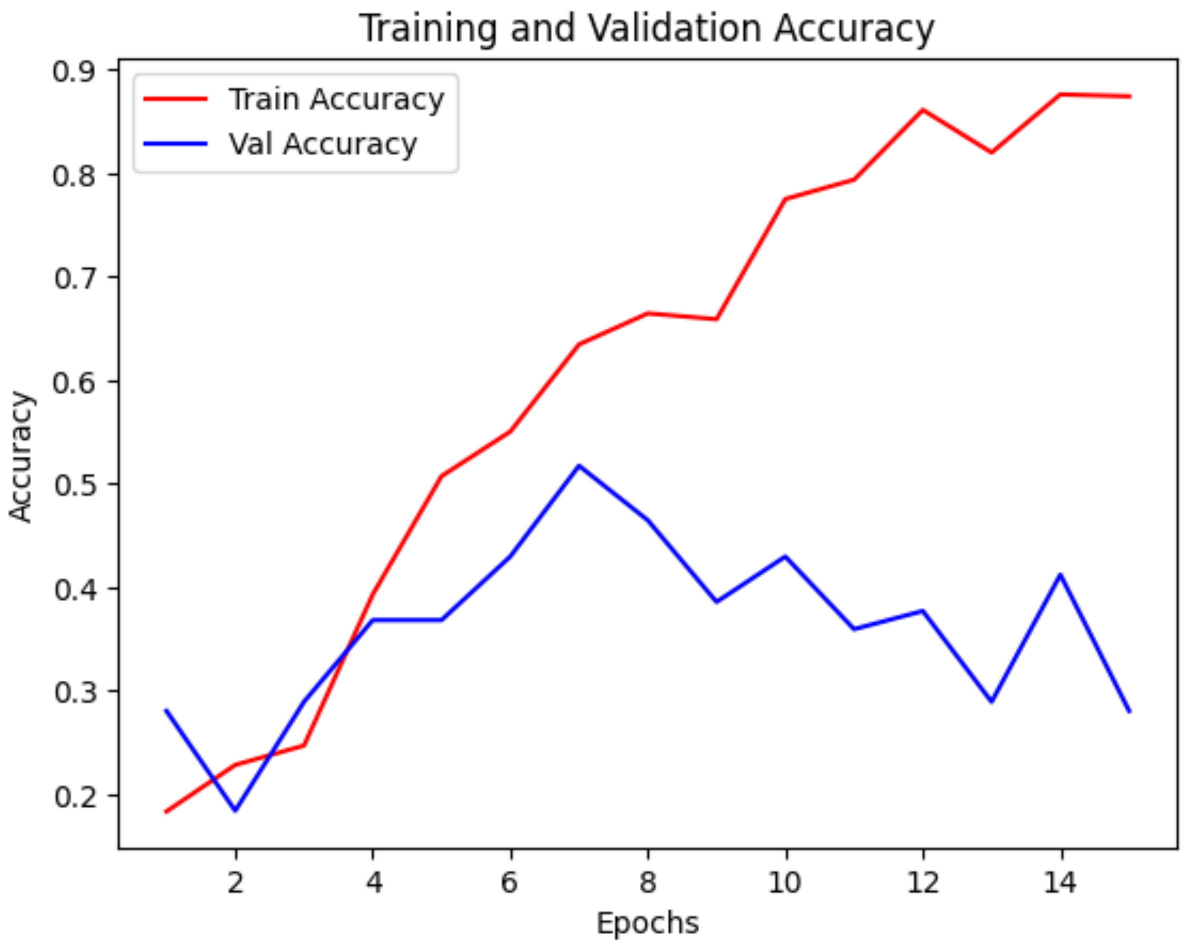
RMSProp

```
inputs = Input(shape=(512, 384, 3))
x = inputs
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dense(64, activation= 'relu', kernel_regularizer= keras.regularizers.L
x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Flatten()(x)
outputs = Dense(6, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)
```

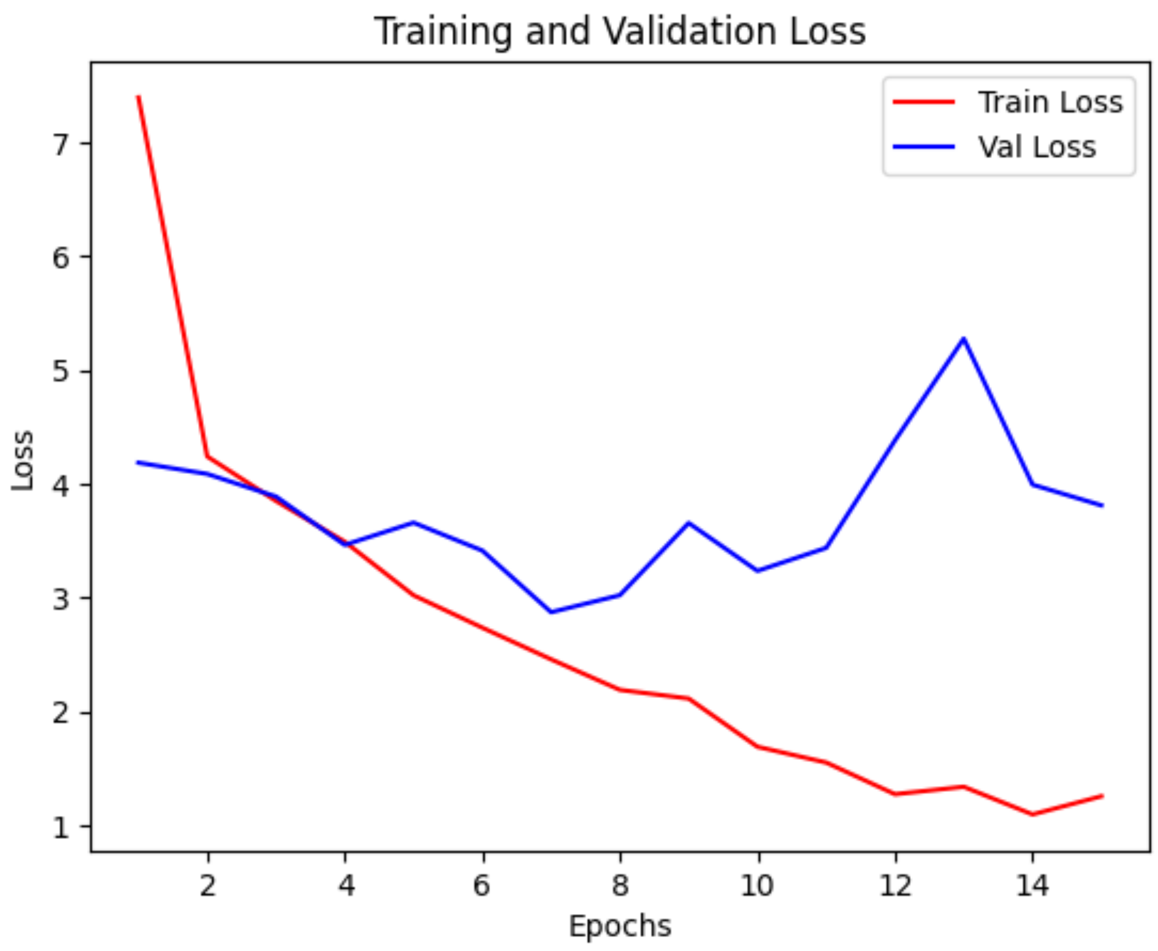
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
history = model.fit(train_dataset, validation_data=validation_dataset, epochs=40,
                    callbacks=[earlystopping])
history_dict = history.history
```

```
Epoch 1/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 378ms/step - accuracy: 0.1435 - loss: 11.0645 -
Epoch 2/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 300ms/step - accuracy: 0.2502 - loss: 4.3739 -
Epoch 3/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 252ms/step - accuracy: 0.2121 - loss: 3.8915 -
Epoch 4/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 271ms/step - accuracy: 0.3633 - loss: 3.5387 -
Epoch 5/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 244ms/step - accuracy: 0.4793 - loss: 3.0938 -
Epoch 6/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 254ms/step - accuracy: 0.5028 - loss: 2.8756 -
Epoch 7/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 275ms/step - accuracy: 0.6371 - loss: 2.4932 -
Epoch 8/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 245ms/step - accuracy: 0.6487 - loss: 2.2363 -
Epoch 9/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 292ms/step - accuracy: 0.7217 - loss: 1.9559 -
Epoch 10/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 267ms/step - accuracy: 0.7633 - loss: 1.7626 -
Epoch 11/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 245ms/step - accuracy: 0.8074 - loss: 1.5632 -
Epoch 12/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 268ms/step - accuracy: 0.8573 - loss: 1.3116 -
Epoch 13/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 261ms/step - accuracy: 0.7981 - loss: 1.4578 -
Epoch 14/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 252ms/step - accuracy: 0.8441 - loss: 1.1844 -
Epoch 15/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 6s 299ms/step - accuracy: 0.9141 - loss: 1.0261 -
```

```
sequence = range(1, len(history_dict['accuracy']) + 1, 1)
plt.plot(sequence, history_dict['accuracy'], 'r', label='Train Accuracy')
plt.plot(sequence, history_dict['val_accuracy'], 'b', label='Val Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



```
plt.plot(sequence, history_dict['loss'], 'r', label='Train Loss')
plt.plot(sequence, history_dict['val_loss'], 'b', label='Val Loss')
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
test_loss, test_accuracy = model.evaluate(test_dataset)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 110ms/step - accuracy: 0.2371 - loss: 4.6765
Test Loss: 4.712001800537109
Test Accuracy: 0.22807016968727112
```

Pretrained Model

```
from tensorflow.keras.applications.resnet50 import ResNet50

resnet_model = ResNet50(input_shape=(512, 384, 3),
                        classes = 6,
                        include_top=False,
                        weights='imagenet')

resnet_model.trainable = False

resnet_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicat
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
Model: "resnet50"
```

Layer (type)	Output Shape	Param #	Conne
input_layer (InputLayer)	(None, 512, 384, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 518, 390, 3)	0	input_
conv1_conv (Conv2D)	(None, 256, 192, 64)	9,472	conv1_
conv1_bn (BatchNormalization)	(None, 256, 192, 64)	256	conv1_
conv1_relu (Activation)	(None, 256, 192, 64)	0	conv1_
pool1_pad (ZeroPadding2D)	(None, 258, 194, 64)	0	conv1_
pool1_pool (MaxPooling2D)	(None, 128, 96, 64)	0	pool1_
conv2_block1_1_conv (Conv2D)	(None, 128, 96, 64)	4,160	pool1_
conv2_block1_1_bn (BatchNormalization)	(None, 128, 96, 64)	256	conv2_
conv2_block1_1_relu (Activation)	(None, 128, 96, 64)	0	conv2_
conv2_block1_2_conv (Conv2D)	(None, 128, 96, 64)	36,928	conv2_
conv2_block1_2_bn	(None, 128, 96, 64)	256	conv2

```
│ (BatchNormalization)      │                        │               │            │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
│ conv2_block1_2_relu       │ (None, 128, 96, 64)    │             0 │ conv2_     │
│ (Activation)              │                        │               │            │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
│ conv2_block1_0_conv       │ (None, 128, 96, 256)   │        16,640 │ pool1_     │
│ (Conv2D)                  │                        │               │            │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
│ conv2_block1_3_conv       │ (None, 128, 96, 256)   │        16,640 │ conv2_     │
│ (Conv2D)                  │                        │               │            │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
│ conv2_block1_0_bn         │ (None, 128, 96, 256)   │         1,024 │ conv2_     │
│ (BatchNormalization)      │                        │               │            │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
│ conv2_block1_3_bn         │ (None, 128, 96, 256)   │         1,024 │ conv2_     │
│ (BatchNormalization)      │                        │               │            │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
│ conv2_block1_add (Add)    │ (None, 128, 96, 256)   │             0 │ conv2_     │
│                           │                        │               │ conv2_     │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
│ conv2_block1_out          │ (None, 128, 96, 256)   │             0 │ conv2_     │
│ (Activation)              │                        │               │            │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
│ conv2_block2_1_conv       │ (None, 128, 96, 64)    │        16,448 │ conv2_     │
│ (Conv2D)                  │                        │               │            │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
│ conv2_block2_1_bn         │ (None, 128, 96, 64)    │           256 │ conv2_     │
│ (BatchNormalization)      │                        │               │            │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
```

```python
def preprocessing(image, label):
    image = tf.keras.applications.resnet50.preprocess_input(image)
    return image, label

train_dataset = train_dataset.map(preprocessing)
validation_dataset = validation_dataset.map(preprocessing)
test_dataset = test_dataset.map(preprocessing)
```

```
│ conv2_block2_2_relu       │ (None, 128, 96, 64)    │             0 │ conv2_     │
│ (Activation)              │                        │               │            │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
│ conv2_block2_3_conv       │ (None, 128, 96, 256)   │        16,640 │ conv2_     │
│ (Conv2D)                  │                        │               │            │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
│ conv2_block2_3_bn         │ (None, 128, 96, 256)   │         1,024 │ conv2_     │
│ (BatchNormalization)      │                        │               │            │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
│ conv2_block2_add (Add)    │ (None, 128, 96, 256)   │             0 │ conv2_     │
│                           │                        │               │ conv2_     │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
│ conv2_block2_out          │ (None, 128, 96, 256)   │             0 │ conv2_     │
│ (Activation)              │                        │               │            │
├───────────────────────────┼────────────────────────┼───────────────┼────────────┤
```

```
import numpy as np


def get_features_and_labels(dataset):
    all_features = []
    all_labels = []
    for images, labels in dataset:
        features = resnet_model.predict(images)
        all_features.append(features)
        all_labels.append(labels)
    return np.concatenate(all_features), np.concatenate(all_labels)

train_features, train_labels = get_features_and_labels(train_dataset)
val_features, val_labels = get_features_and_labels(validation_dataset)
test_features, test_labels = get_features_and_labels(test_dataset)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 189ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 198ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 202ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 273ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 547ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 271ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 367ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 316ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 384ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 412ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 269ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 322ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 238ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 372ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 160ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 172ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 169ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 213ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 129ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 145ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
```

```
train_features.shape
```

```
(534, 16, 12, 2048)
```

(Conv2D)

Start coding or generate with AI.

Optimizers

```
conv3
```

SGD

```
inputs = layers.Input(shape=(16, 12, 2048))
x = layers.Flatten()(inputs)
outputs = layers.Dense(6, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)

model.compile(optimizer='sgd',
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])
```

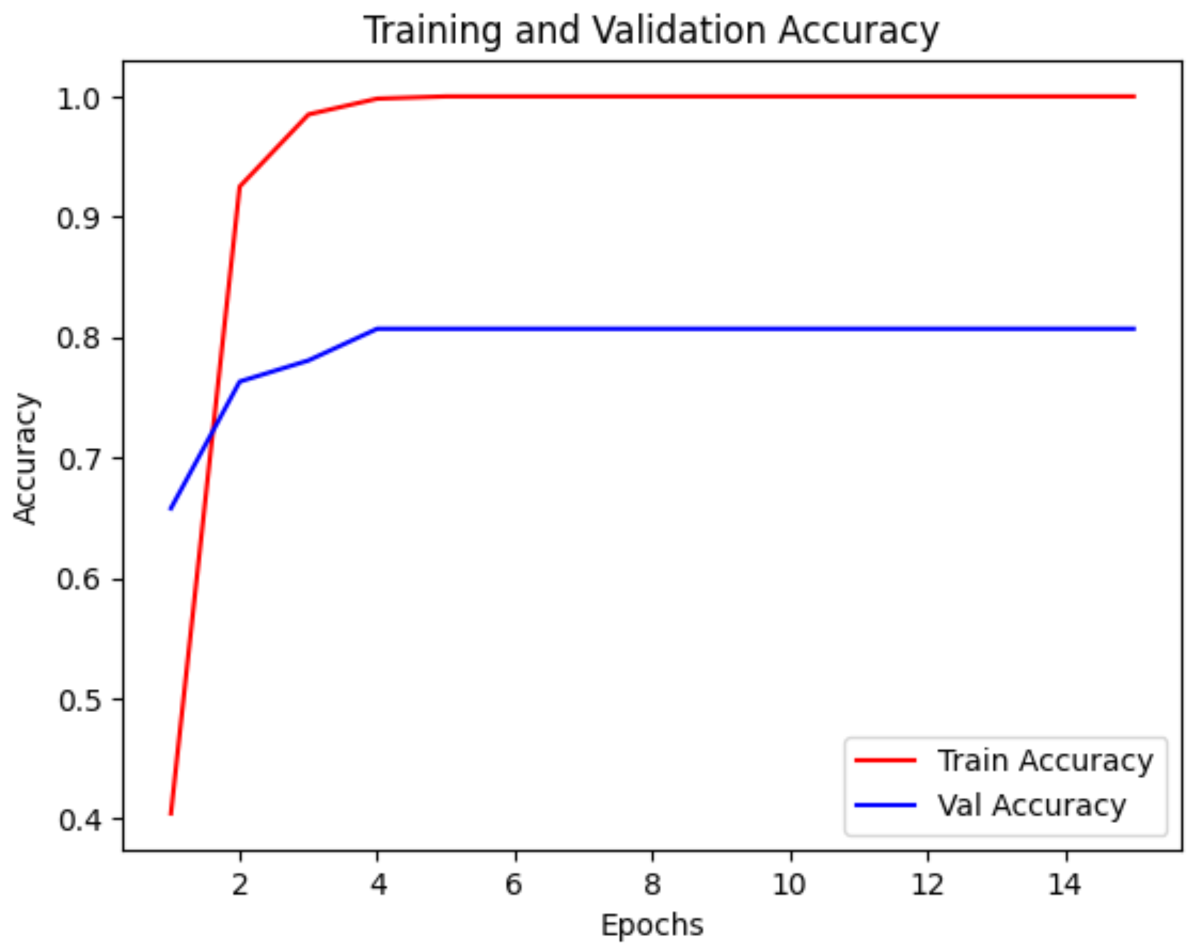
```
│ conv3_block2_2_bn           │ (None, 64, 48, 128)      │          512 │ conv3_
│ (BatchNormalization)        │                          │              │
├─────────────────────────────┼──────────────────────────┼──────────────┼───────
│ conv3_block2_2_relu         │ (None, 64, 48, 128)      │            0 │ conv3_
│ (Activation)                │                          │              │
├─────────────────────────────┼──────────────────────────┼──────────────┼───────
│ conv3_block2_3_conv         │ (None, 64, 48, 512)      │       66,048 │ conv3_
│ (Conv2D)                    │                          │              │
├─────────────────────────────┼──────────────────────────┼──────────────┼───────
│ conv3_block2_3_bn           │ (None, 64, 48, 512)      │        2,048 │ conv3_
│ (BatchNormalization)        │                          │              │
├─────────────────────────────┼──────────────────────────┼──────────────┼───────
│ conv3_block2_add (Add)      │ (None, 64, 48, 512)      │            0 │ conv3_
│                             │                          │              │ conv3_
├─────────────────────────────┼──────────────────────────┼──────────────┼───────
│ conv3_block2_out            │ (None, 64, 48, 512)      │            0 │ conv3_
│ (Activation)                │                          │              │
├─────────────────────────────┼──────────────────────────┼──────────────┼───────
│ conv3_block3_1_conv         │ (None, 64, 48, 128)      │       65,664 │ conv3_
│ (Conv2D)                    │                          │              │
├─────────────────────────────┼──────────────────────────┼──────────────┼───────
│ conv3_block3_1_bn           │ (None, 64, 48, 128)      │          512 │ conv3_
│ (BatchNormalization)        │                          │              │
├─────────────────────────────┼──────────────────────────┼──────────────┼───────
│ conv3_block3_1_relu         │ (None, 64, 48, 128)      │            0 │ conv3_
│ (Activation)                │                          │              │
├─────────────────────────────┼──────────────────────────┼──────────────┼───────
│ conv3_block3_2_conv         │ (None, 64, 48, 128)      │      147,584 │ conv3_
│ (Conv2D)                    │                          │              │
```

```
history = model.fit(train_features, train_labels,
                    epochs=40, batch_size = 32,
                    validation_data=(val_features, val_labels),
                    callbacks= [earlystopping]
                    )

history_dict = history.history
```

```
Epoch 1/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 2s 122ms/step - accuracy: 0.3173 - loss: 186.3167 -
Epoch 2/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9230 - loss: 9.5114 - va
Epoch 3/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9849 - loss: 0.9588 - va
Epoch 4/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9965 - loss: 0.0041 - va
Epoch 5/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 1.0000 - loss: 1.9797e-08
Epoch 6/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 1.0000 - loss: 1.4534e-08
Epoch 7/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 1.0000 - loss: 2.2351e-08
Epoch 8/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 2s 67ms/step - accuracy: 1.0000 - loss: 4.9701e-08
Epoch 9/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 1.0000 - loss: 2.1304e-08
Epoch 10/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 1.0000 - loss: 2.1057e-08
Epoch 11/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 1.0000 - loss: 1.4779e-08
Epoch 12/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 1.0000 - loss: 5.2271e-09
Epoch 13/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 1.0000 - loss: 1.9506e-08
Epoch 14/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 1.0000 - loss: 3.5196e-08
Epoch 15/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 1.0000 - loss: 3.1682e-09
```

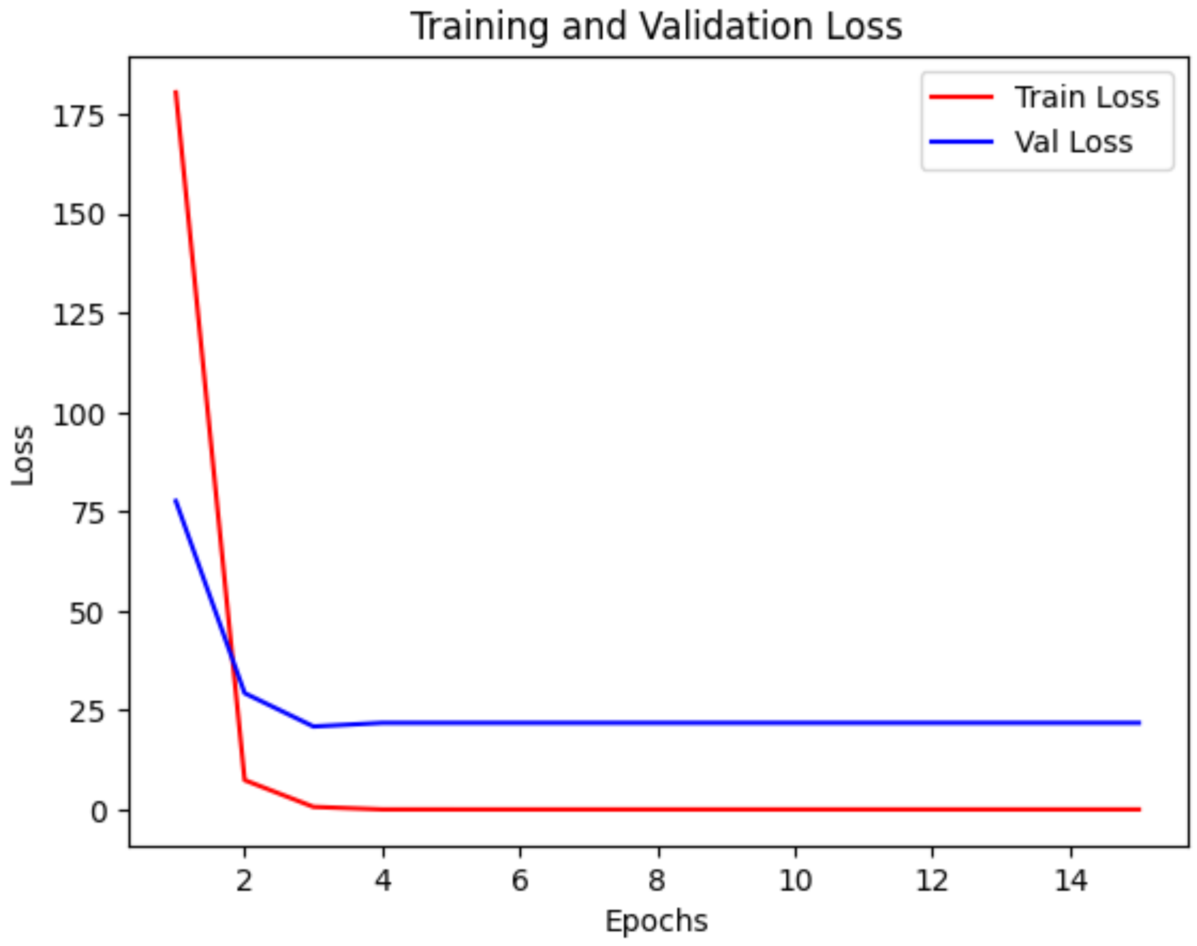
conv3_block4_add (Add)	(None, 64, 48, 512)	0	conv3_ conv3_
conv3_block4_out (Activation)	(None, 64, 48, 512)	0	conv3_
conv4_block1_1_conv (Conv2D)	(None, 32, 24, 256)	131,328	conv3_
conv4_block1_1_bn (BatchNormalization)	(None, 32, 24, 256)	1,024	conv4_

```
sequence = range(1, len(history_dict['accuracy']) + 1, 1)
plt.plot(sequence, history_dict['accuracy'], 'r', label='Train Accuracy')
plt.plot(sequence, history_dict['val_accuracy'], 'b', label='Val Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



conv4_block2_2_conv (Conv2D)	(None, 32, 24, 256)	590,080	conv4_
conv4_block2_2_bn (BatchNormalization)	(None, 32, 24, 256)	1,024	conv4_
conv4_block2_2_relu (Activation)	(None, 32, 24, 256)	0	conv4_
conv4_block2_3_conv (Conv2D)	(None, 32, 24, 1024)	263,168	conv4_
conv4_block2_3_bn (BatchNormalization)	(None, 32, 24, 1024)	4,096	conv4_

```
plt.plot(sequence, history_dict['loss'], 'r', label='Train Loss')
plt.plot(sequence, history_dict['val_loss'], 'b', label='Val Loss')
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
test_loss, test_accuracy = model.evaluate(test_features, test_labels)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.6824 - loss: 33.6395
Test Loss: 32.708919525146484
Test Accuracy: 0.6929824352264404
```

(BatchNormalization)			
conv4_block4_2_relu	(None, 32, 24, 256)	0	conv4_

Adam

```
│ (Conv2D)                     │                           │                 │

inputs = layers.Input(shape=(16, 12, 2048))
x = layers.Flatten()(inputs)
outputs = layers.Dense(6, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)


model.compile(optimizer='adam',
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])
```

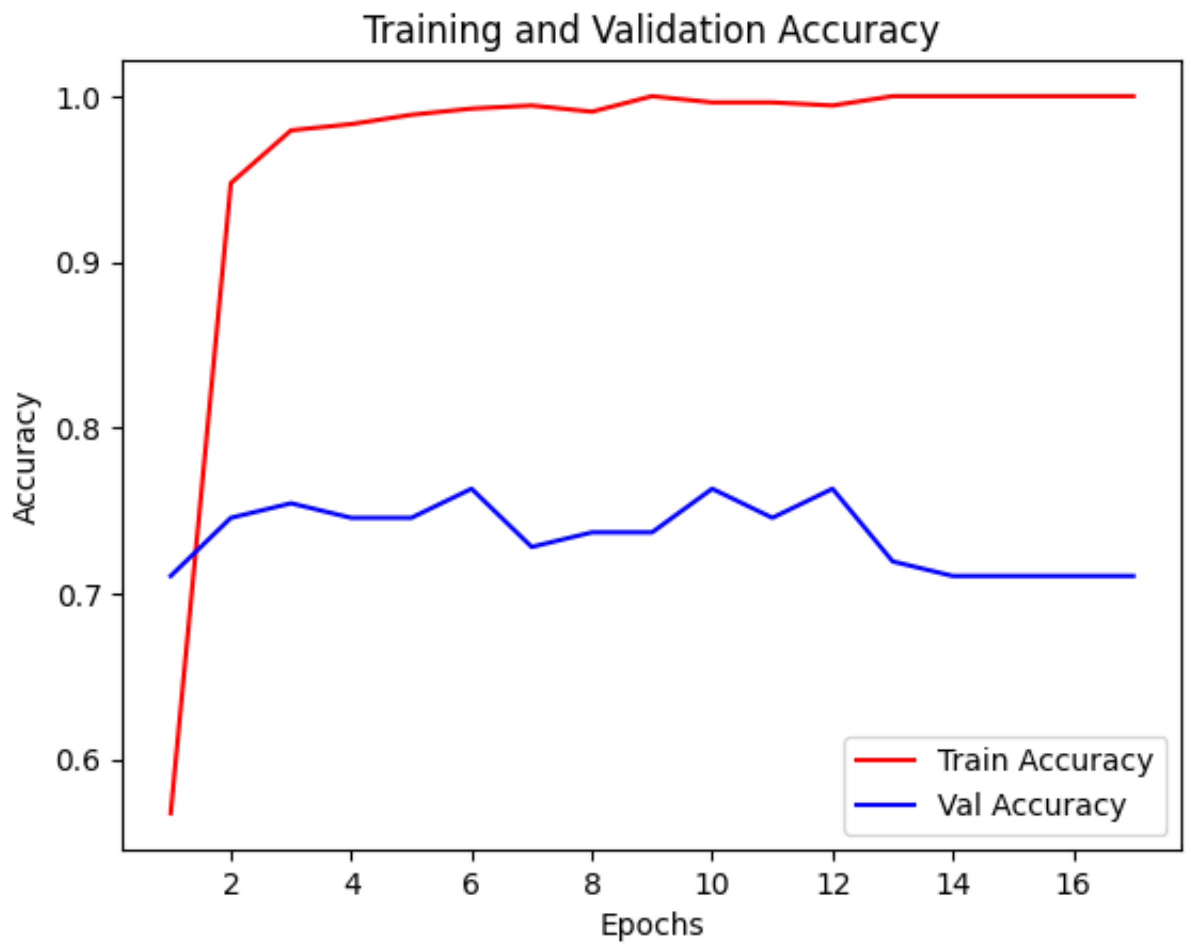
```
│ conv4_block5_1_bn            │ (None, 32, 24, 256)       │           1,024 │ conv4_
│ (BatchNormalization)         │                           │                 │
├──────────────────────────────┼───────────────────────────┼─────────────────┼───────
│ conv4_block5_1_relu          │ (None, 32, 24, 256)       │               0 │ conv4_
│ (Activation)                 │                           │                 │
├──────────────────────────────┼───────────────────────────┼─────────────────┼───────
│ conv4_block5_2_conv          │ (None, 32, 24, 256)       │         590,080 │ conv4_
│ (Conv2D)                     │                           │                 │
├──────────────────────────────┼───────────────────────────┼─────────────────┼───────
│ conv4_block5_2_bn            │ (None, 32, 24, 256)       │           1,024 │ conv4_
│ (BatchNormalization)         │                           │                 │
├──────────────────────────────┼───────────────────────────┼─────────────────┼───────
│ conv4_block5_2_relu          │ (None, 32, 24, 256)       │               0 │ conv4_
│ (Activation)                 │                           │                 │
├──────────────────────────────┼───────────────────────────┼─────────────────┼───────
│ conv4_block5_3_conv          │ (None, 32, 24, 1024)      │         263,168 │ conv4_
│ (Conv2D)                     │                           │                 │
├──────────────────────────────┼───────────────────────────┼─────────────────┼───────
│ conv4_block5_3_bn            │ (None, 32, 24, 1024)      │           4,096 │ conv4_
│ (BatchNormalization)         │                           │                 │
├──────────────────────────────┼───────────────────────────┼─────────────────┼───────
│ conv4_block5_add (Add)       │ (None, 32, 24, 1024)      │               0 │ conv4_
│                              │                           │                 │ conv4_
├──────────────────────────────┼───────────────────────────┼─────────────────┼───────
│ conv4_block5_out             │ (None, 32, 24, 1024)      │               0 │ conv4_
│ (Activation)                 │                           │                 │
├──────────────────────────────┼───────────────────────────┼─────────────────┼───────
│ conv4_block6_1_conv          │ (None, 32, 24, 256)       │         262,400 │ conv4_
│ (Conv2D)                     │                           │                 │
├──────────────────────────────┼───────────────────────────┼─────────────────┼───────
│ conv4_block6_1_bn            │ (None, 32, 24, 256)       │           1,024 │ conv4_
│ (BatchNormalization)         │                           │                 │
├──────────────────────────────┼───────────────────────────┼─────────────────┼───────
│ conv4_block6_1_relu          │ (None, 32, 24, 256)       │               0 │ conv4_
│ (Activation)                 │                           │                 │
├──────────────────────────────┼───────────────────────────┼─────────────────┼───────
│ conv4_block6_2_conv          │ (None, 32, 24, 256)       │         590,080 │ conv4_
```

```
history = model.fit(train_features, train_labels,
                    epochs=40, batch_size = 32,
                    validation_data=(val_features, val_labels),
                    callbacks= [earlystopping]
                    )

history_dict = history.history
```

```
Epoch 1/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 3s 165ms/step - accuracy: 0.4371 - loss: 19.4941 -
Epoch 2/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 3s 39ms/step - accuracy: 0.9523 - loss: 1.7503 - va
Epoch 3/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9820 - loss: 0.2692 - va
Epoch 4/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9856 - loss: 0.3582 - va
Epoch 5/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9900 - loss: 0.1556 - va
Epoch 6/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9919 - loss: 0.1146 - va
Epoch 7/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9981 - loss: 0.0167 - va
Epoch 8/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9922 - loss: 0.0494 - va
Epoch 9/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 1.0000 - loss: 6.6730e-07
Epoch 10/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9981 - loss: 0.0060 - va
Epoch 11/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 57ms/step - accuracy: 0.9991 - loss: 0.0252 - va
Epoch 12/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - accuracy: 0.9975 - loss: 0.0121 - va
Epoch 13/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 1.0000 - loss: 1.7688e-04
Epoch 14/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 1.0000 - loss: 1.6995e-07
Epoch 15/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 1.0000 - loss: 8.6398e-07
Epoch 16/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 1.0000 - loss: 2.4295e-06
Epoch 17/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 1.0000 - loss: 2.3312e-06
```

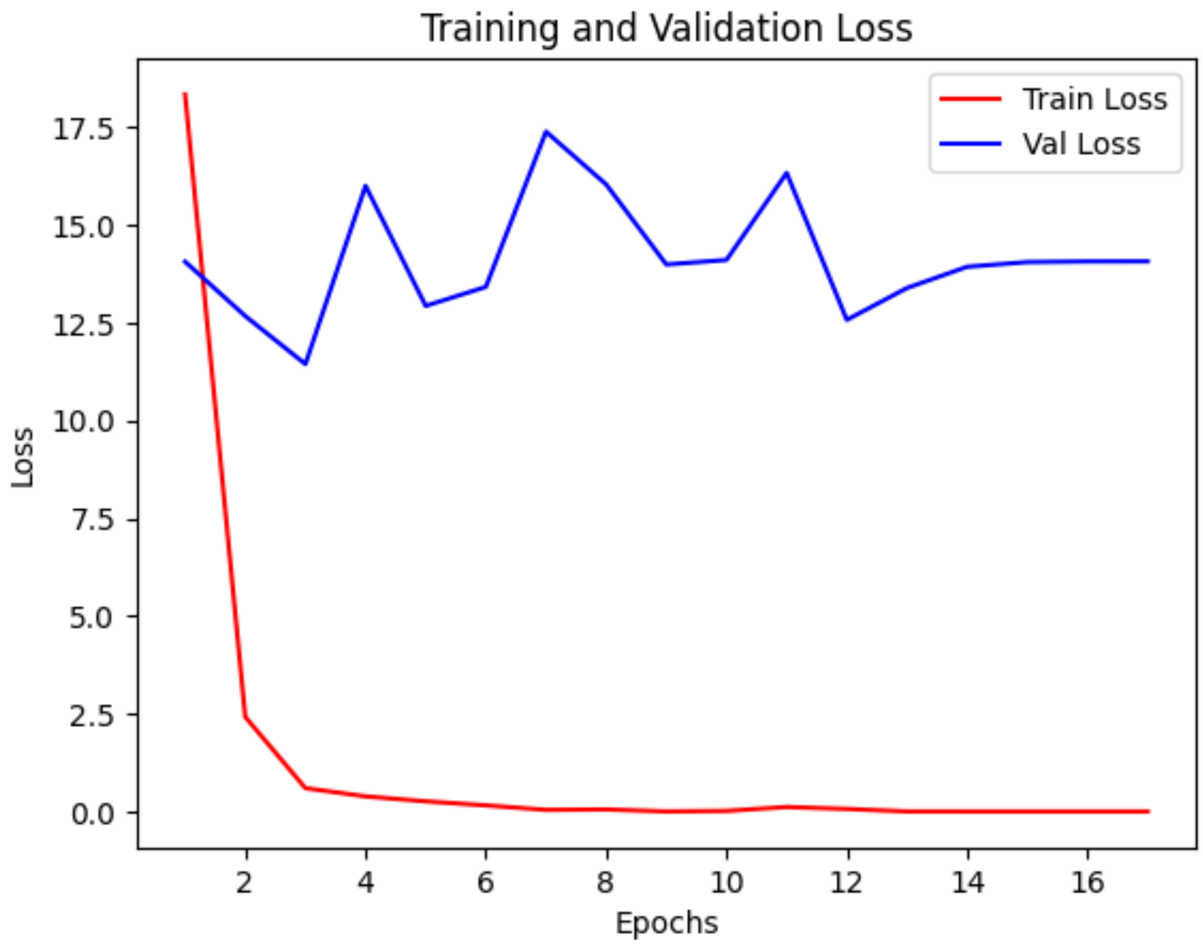
conv5_block1_3_bn (BatchNormalization)	(None, 16, 12, 2048)	8,192	conv5_
conv5_block1_add (Add)	(None, 16, 12, 2048)	0	conv5_ conv5_

```
sequence = range(1, len(history_dict['accuracy']) + 1, 1)
plt.plot(sequence, history_dict['accuracy'], 'r', label='Train Accuracy')
plt.plot(sequence, history_dict['val_accuracy'], 'b', label='Val Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



(Activation)			
conv5_block3_2_conv (Conv2D)	(None, 16, 12, 512)	2,359,808	conv5_
conv5_block3_2_bn (BatchNormalization)	(None, 16, 12, 512)	2,048	conv5_
conv5_block3_2_relu (Activation)	(None, 16, 12, 512)	0	conv5_
conv5_block3_3_conv (Conv2D)	(None, 16, 12, 2048)	1,050,624	conv5_

```
plt.plot(sequence, history_dict['loss'], 'r', label='Train Loss')
plt.plot(sequence, history_dict['val_loss'], 'b', label='Val Loss')
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
test_loss, test_accuracy = model.evaluate(test_features, test_labels)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.6677 - loss: 20.4179
Test Loss: 21.33689308166504
Test Accuracy: 0.6666666865348816
```

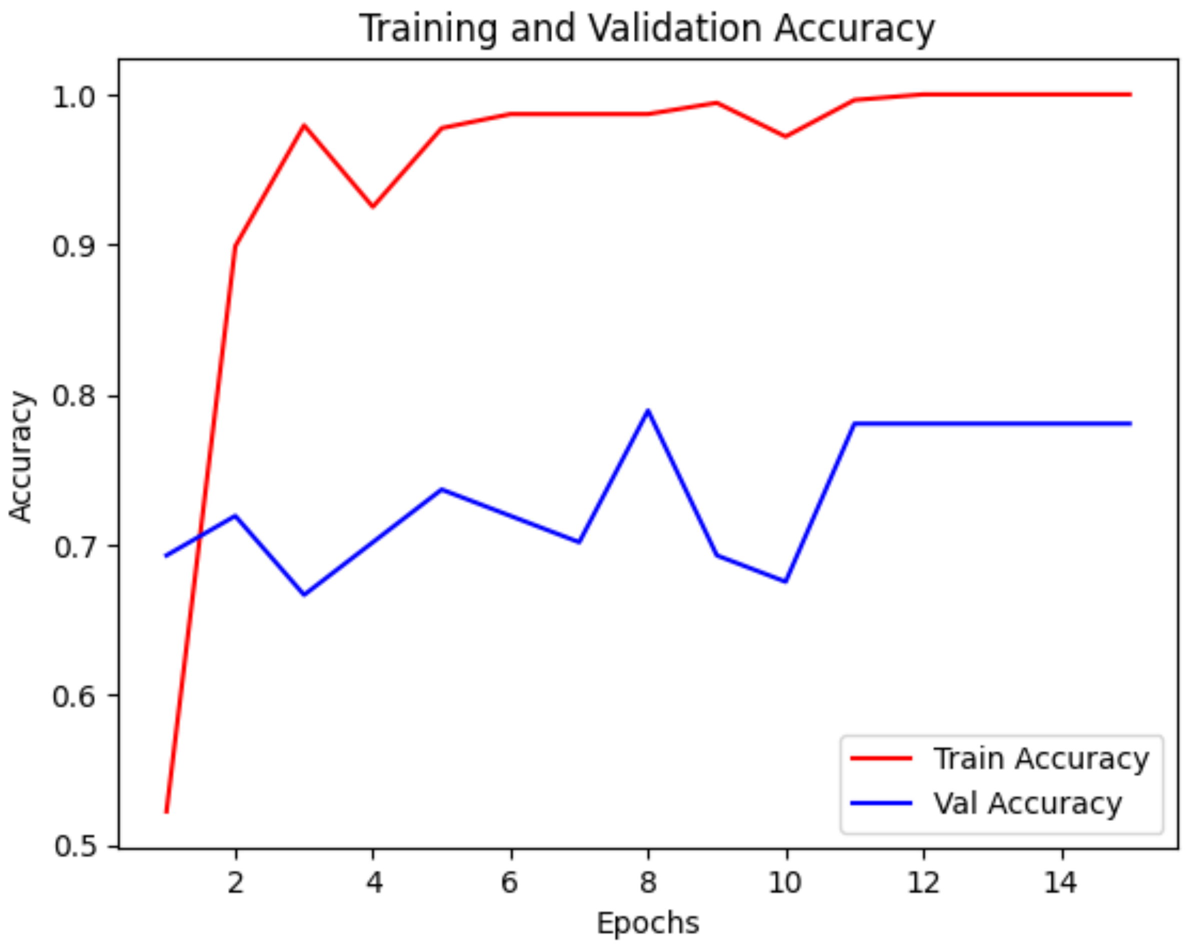
RMSProp

```
inputs = layers.Input(shape=(16, 12, 2048))
x = layers.Flatten()(inputs)
outputs = layers.Dense(6, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)

model.compile(optimizer='rmsprop',
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])
```

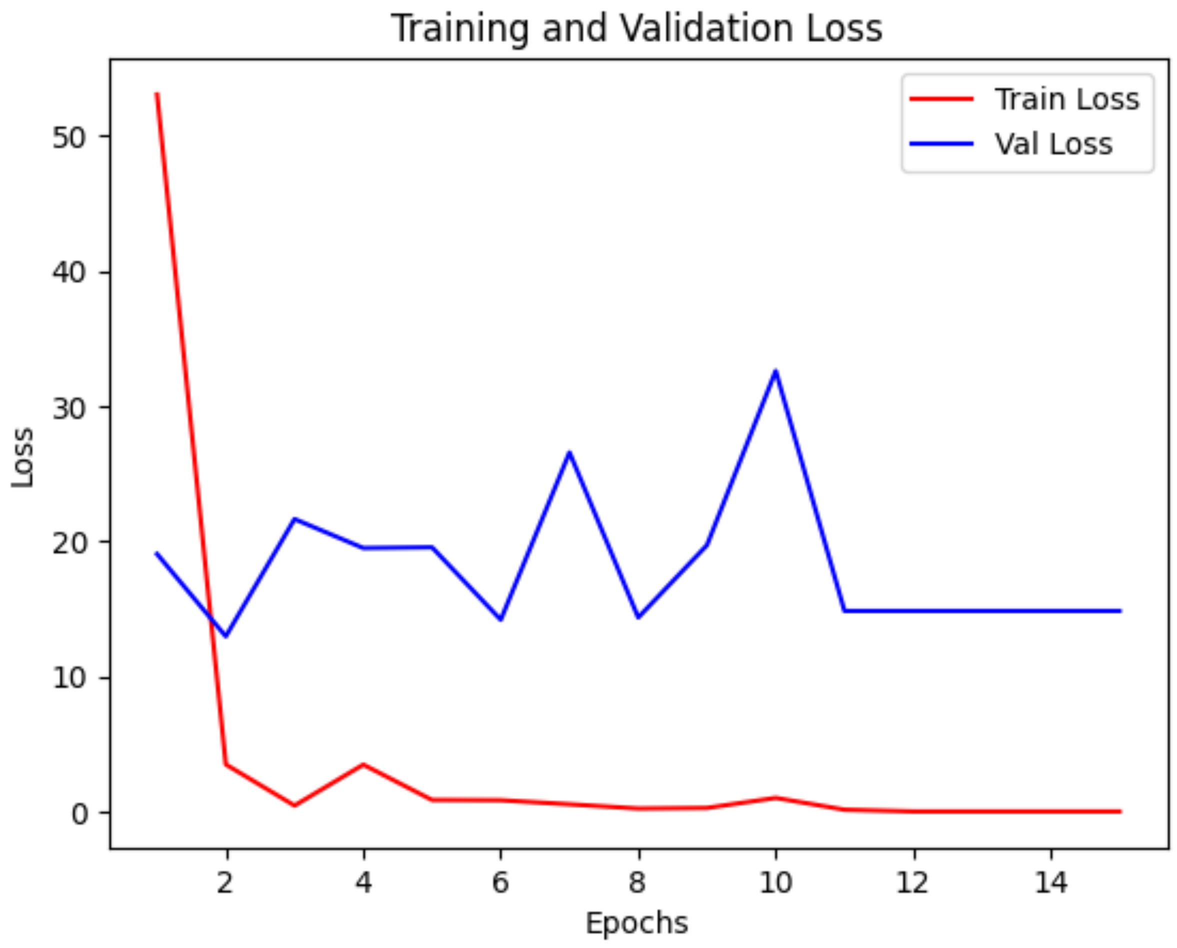
```
history = model.fit(train_features, train_labels,
                    epochs=40, batch_size = 32,
                    validation_data=(val_features, val_labels),
                    callbacks= [earlystopping]
                    )

history_dict = history.history
```

```
Epoch 1/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 3s 121ms/step - accuracy: 0.4009 - loss: 66.7768 -
Epoch 2/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9028 - loss: 3.5241 - va
Epoch 3/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - accuracy: 0.9816 - loss: 0.4046 - va
Epoch 4/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - accuracy: 0.9120 - loss: 3.5203 - va
Epoch 5/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 0.9704 - loss: 0.9387 - va
Epoch 6/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.9880 - loss: 0.7834 - va
Epoch 7/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9869 - loss: 0.4584 - va
Epoch 8/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9828 - loss: 0.2429 - va
Epoch 9/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 53ms/step - accuracy: 0.9985 - loss: 0.0498 - va
Epoch 10/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9864 - loss: 0.4174 - va
Epoch 11/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9928 - loss: 0.2344 - va
Epoch 12/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 1.0000 - loss: 6.5500e-09
Epoch 13/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 1.0000 - loss: 8.6800e-10
Epoch 14/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 1.0000 - loss: 2.6409e-09
Epoch 15/40
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 1.0000 - loss: 4.6930e-09
```

```
sequence = range(1, len(history_dict['accuracy']) + 1, 1)
plt.plot(sequence, history_dict['accuracy'], 'r', label='Train Accuracy')
plt.plot(sequence, history_dict['val_accuracy'], 'b', label='Val Accuracy')
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



```
plt.plot(sequence, history_dict['loss'], 'r', label='Train Loss')
plt.plot(sequence, history_dict['val_loss'], 'b', label='Val Loss')
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



```
test_loss, test_accuracy = model.evaluate(test_features, test_labels)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7242 - loss: 21.7546
Test Loss: 21.755510330200195
Test Accuracy: 0.719298243522644
```