rueegg_wissiak_optimized

March 7, 2023

1 Optimized Model

This model does not show under- or overfitting and performs well on both, training and testing data. Afterwards, a brief description on how to tackle the challenges of an optimal model complexity.

To address underfitting, one approach is to increase the complexity of the model by adding more layers or increasing the number of filters in each layer. To address overfitting, we can try several approaches. One approach is to simplify the model by removing some layers or decreasing the number of filters in each layer. Another approach is to use less epochs for example.

Adding dropout or weight decay can help to address both of the above mentioned issues. We can also try adjusting the hyperparameters such as learning rate, batch size, or number of epochs.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
from tensorflow.keras.utils import image_dataset_from_directory

img_size = 150
batch_size = 64
seed = 31

train_ds = image_dataset_from_directory(
    './dataset/seg_train/seg_train',
    validation_split=0.2,
    subset="training",
    labels="inferred",
    seed=seed,
    image_size=(img_size, img_size),
    batch_size=batch_size
)
val_ds = image_dataset_from_directory(
    './dataset/seg_train/seg_train',
    validation_split=0.2,
    subset="validation",
    labels="inferred",
    seed=seed,
    image_size=(img_size, img_size),
    batch_size=batch_size
```

```
)
test_ds = image_dataset_from_directory(
    './dataset/seg_test/seg_test',
    labels="inferred",
    seed=seed,
    image_size=(img_size, img_size),
    batch_size=batch_size
)
```

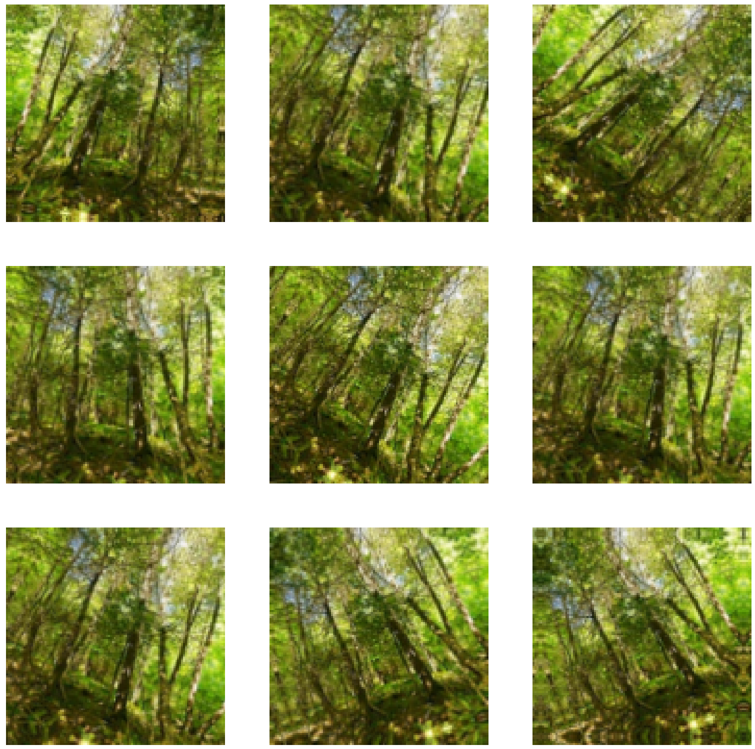
```
Found 2666 files belonging to 5 classes.
Using 2133 files for training.
```

```
2023-03-07 08:10:14.414660: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

```
Found 2666 files belonging to 5 classes.
Using 533 files for validation.
Found 2499 files belonging to 5 classes.
```

[3]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import RandomRotation, RandomZoom, Rescaling,␣
 ↪RandomFlip

data_augmentation = Sequential(
  [
    RandomFlip("horizontal", input_shape=(img_size, img_size, 3)),
    RandomRotation(0.1),
    RandomZoom(0.1),
  ]
)
```

[4]:
```
plt.figure(figsize=(5, 5))
for image, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(image)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```



1.1 Building the Model

Here we use the same model as the overfitting one but we add some extras to reduce the overfitting behavior.

1.1.1 Regularization

Regularization is used to reduce the impact impact of the weights. The weights then end up having less impact on the loss function which determines the error between the actual label and predicted label. This reduces complexity of the model and therefore reduces overfitting. We are adding regularization only to those layers which have the largest number of parameters according to the model summary. We are using L2 (Ridge) regularization since it predetermined from the task. We are adding L2 mainly to the layers that add the most parameters to the CNN.

Dropout Layers: The benefit of using dropout is no node in the network will be assigned with high parameter values, as a a result the parameter values will be dispersed and the output of the current layer will not depend on a single node. E.g. Dropout(0.2) drops the input layers at a probability of 0.2.

1.1.2 Generalization

To improve generalization of the model, data augmentation is a useful tool. With data augmentation we can add artificial effects to the images such as shearing, stretching, flipping, rotating and translating. Through these effects, the images always appear differently each time they appear in the training step and therefore the CNN doesn't adapt to the exact images but rather learns about the relative features inside of an image.

1.1.3 Optimizer

For the optimized model we chose Adam over the competitors because it is the most common among SGD. We tried out SGD but it performed very poorly compared to Adam which might be due to insufficient configuration of the learning rate. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order (mean) and second-order (uncentered variance) moments. Its default implementation already provides a form of annealed learning, beta_1=0.9 for the first-order moment and beta_2=0.999 for the second-order moment.

1.1.4 Activation Function

The following article states that ReLU is the overall the best suited activation function so based on this we decided to use ReLU for our optimized model.

1.1.5 Batch Size

The batch size defines how many samples (images here) run through the Neural Network before the weights get adapted. It is recommended to use mini batches to update the Neural Network multiple times during an epoch. We've tried out diffrent batch sizes with the same seed on the image generator TODO

[5]:
```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout
from tensorflow.keras.regularizers import l2
```

[6]:
```
def create_model(l2_param=0.001, last_dropout_param=0.2):
    model = Sequential()

    model.add(data_augmentation)
    model.add(Rescaling(1./255))

    model.add(Conv2D(32, (3,3), input_shape= (img_size,img_size,3), activation␣
↪= 'relu', padding = 'same')) #padding = same size output
    model.add(MaxPooling2D())

    model.add(Conv2D(64, (3,3), activation = 'relu', padding = 'same'))
    model.add(MaxPooling2D())

    model.add(Conv2D(128, (3,3), activation = 'relu', padding = 'same'))
    model.add(MaxPooling2D())

    model.add(Conv2D(256, (3,3), activation = 'relu', padding = 'same'))
    model.add(MaxPooling2D())

    model.add(Conv2D(512, (3,3), activation = 'relu', padding = 'same',␣
↪kernel_regularizer=l2(l=l2_param)))
```

```
    model.add(MaxPooling2D())
    model.add(Dropout(0.1))

    model.add(Conv2D(1024, (3,3), activation = 'relu', padding = 'same',␣
↪kernel_regularizer=l2(l=l2_param)))
    model.add(MaxPooling2D())
    model.add(Dropout(0.1))

    model.add(Conv2D(512, (3,3), activation = 'relu', padding = 'same',␣
↪kernel_regularizer=l2(l=l2_param)))
    model.add(MaxPooling2D())

    model.add(Dropout(last_dropout_param))
    model.add(Flatten())

    model.add(Dense(256, activation='relu'))
    #model.add(Dense(512, activation='relu'))
    model.add(Dense(5, activation = 'softmax'))

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

    return model
```

1.2 Cross Validation for L2 Parameter

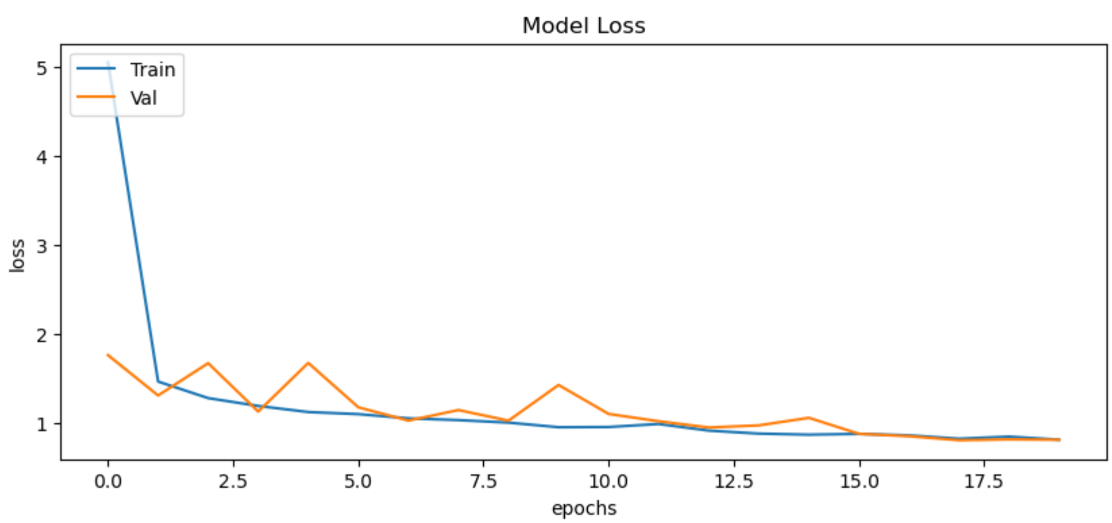
To find the optimal L2 regularization parameter we are using GridSearchCV from sklearn by applying a k=5 Cross Validation. As the to-be-optimized score we use, as per default, the accuracy. To be able to use GridSearchCV we must wrap the model to be compatible with the sklearn ecosystem.

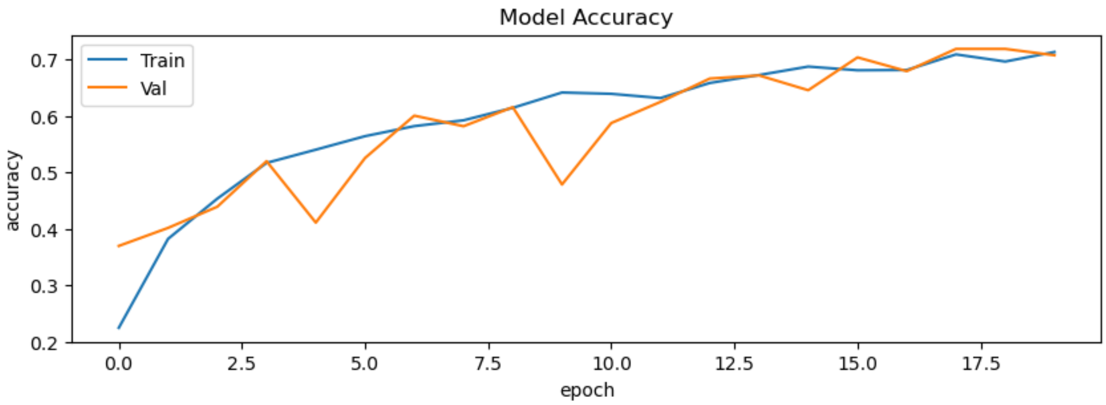
[7]:
```
import itertools
param_grid=dict(
    #l2_param=[0.01, 0.001, 0.0001],
    last_dropout_param=[0.1, 0.2, 0.3],
    #batch_size_param=[16, 32, 128],
    #epochs_param=[10, 20, 30]
)

keys = list(param_grid.keys())
params = list(param_grid.get(x) for x in keys)
param_permutations = list(itertools.product(*params))

model_history = []

for perm in param_permutations:
    model_args = dict()
    for index in range(len(params)):
```

```
        key = keys[index]
        value = perm[index]
        model_args[key] = value
    print("CURRENTLY TRAINING THE MODEL WITH THE FOLLOWING PARAMS:")
    print(model_args)
    model = create_model(**model_args)
    history = model.fit(train_ds, validation_data=val_ds, epochs=20)
    %run rueegg_wissiak_model_visualization.ipynb
    eval = model.evaluate(test_ds)
    print(eval)
    model_history.append(dict(history=history, model=model))
```

```
CURRENTLY TRAINING THE MODEL WITH THE FOLLOWING PARAMS:
{'l2_param': 0.01}
Epoch 1/20
34/34 [==============================] - 76s 2s/step - loss: 5.0521 - accuracy:
0.2246 - val_loss: 1.7668 - val_accuracy: 0.3696
Epoch 2/20
34/34 [==============================] - 74s 2s/step - loss: 1.4704 - accuracy:
0.3821 - val_loss: 1.3141 - val_accuracy: 0.4015
Epoch 3/20
34/34 [==============================] - 75s 2s/step - loss: 1.2844 - accuracy:
0.4534 - val_loss: 1.6768 - val_accuracy: 0.4390
Epoch 4/20
34/34 [==============================] - 74s 2s/step - loss: 1.1967 - accuracy:
0.5166 - val_loss: 1.1333 - val_accuracy: 0.5197
Epoch 5/20
34/34 [==============================] - 74s 2s/step - loss: 1.1282 - accuracy:
0.5401 - val_loss: 1.6796 - val_accuracy: 0.4109
Epoch 6/20
34/34 [==============================] - 75s 2s/step - loss: 1.1046 - accuracy:
0.5640 - val_loss: 1.1814 - val_accuracy: 0.5253
Epoch 7/20
34/34 [==============================] - 75s 2s/step - loss: 1.0573 - accuracy:
0.5818 - val_loss: 1.0321 - val_accuracy: 0.6004
Epoch 8/20
34/34 [==============================] - 75s 2s/step - loss: 1.0377 - accuracy:
0.5921 - val_loss: 1.1506 - val_accuracy: 0.5816
Epoch 9/20
34/34 [==============================] - 75s 2s/step - loss: 1.0094 - accuracy:
0.6142 - val_loss: 1.0313 - val_accuracy: 0.6154
Epoch 10/20
34/34 [==============================] - 75s 2s/step - loss: 0.9579 - accuracy:
0.6414 - val_loss: 1.4322 - val_accuracy: 0.4784
Epoch 11/20
34/34 [==============================] - 75s 2s/step - loss: 0.9602 - accuracy:
0.6390 - val_loss: 1.1070 - val_accuracy: 0.5872
```

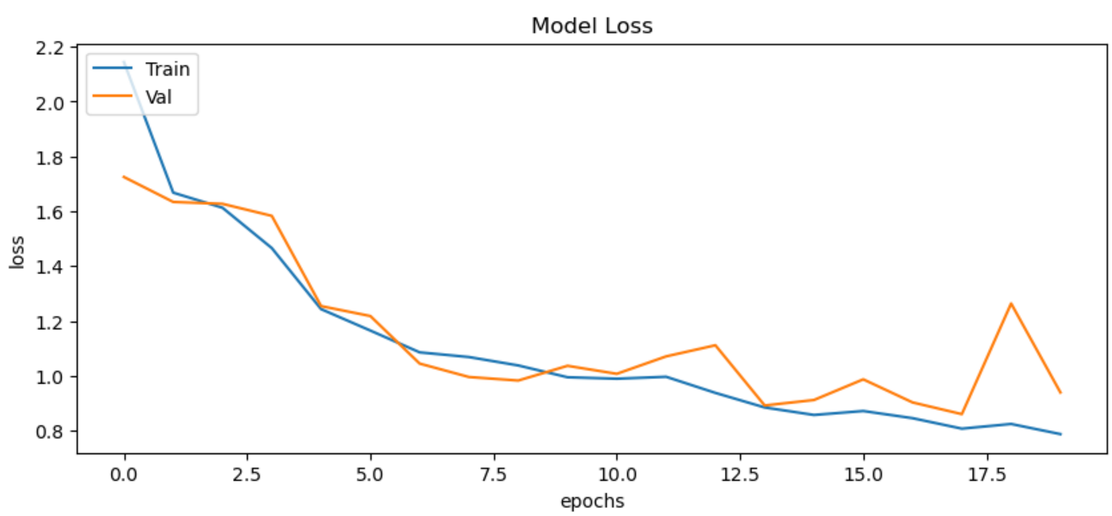
```
Epoch 12/20
34/34 [==============================] - 75s 2s/step - loss: 0.9940 - accuracy:
0.6315 - val_loss: 1.0248 - val_accuracy: 0.6248
Epoch 13/20
34/34 [==============================] - 75s 2s/step - loss: 0.9199 - accuracy:
0.6582 - val_loss: 0.9544 - val_accuracy: 0.6660
Epoch 14/20
34/34 [==============================] - 75s 2s/step - loss: 0.8861 - accuracy:
0.6723 - val_loss: 0.9782 - val_accuracy: 0.6717
Epoch 15/20
34/34 [==============================] - 75s 2s/step - loss: 0.8743 - accuracy:
0.6873 - val_loss: 1.0632 - val_accuracy: 0.6454
Epoch 16/20
34/34 [==============================] - 75s 2s/step - loss: 0.8830 - accuracy:
0.6807 - val_loss: 0.8839 - val_accuracy: 0.7036
Epoch 17/20
34/34 [==============================] - 75s 2s/step - loss: 0.8666 - accuracy:
0.6812 - val_loss: 0.8567 - val_accuracy: 0.6792
Epoch 18/20
34/34 [==============================] - 75s 2s/step - loss: 0.8289 - accuracy:
0.7089 - val_loss: 0.8118 - val_accuracy: 0.7186
Epoch 19/20
34/34 [==============================] - 75s 2s/step - loss: 0.8523 - accuracy:
0.6962 - val_loss: 0.8210 - val_accuracy: 0.7186
Epoch 20/20
34/34 [==============================] - 75s 2s/step - loss: 0.8179 - accuracy:
0.7131 - val_loss: 0.8169 - val_accuracy: 0.7073
```

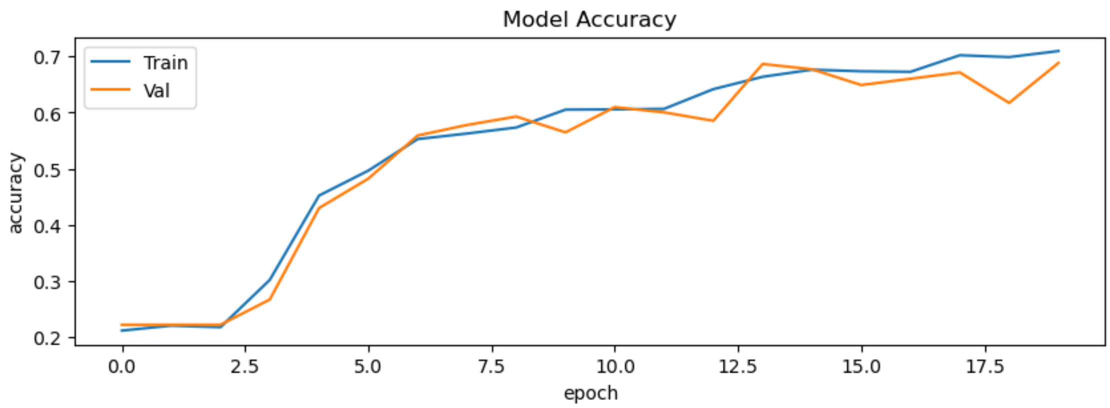





```
40/40 [==============================] - 13s 329ms/step - loss: 0.7990 -
accuracy: 0.7167
CURRENTLY TRAINING THE MODEL WITH THE FOLLOWING PARAMS:
{'l2_param': 0.001}
Epoch 1/20
34/34 [==============================] - 76s 2s/step - loss: 2.1442 - accuracy:
0.2110 - val_loss: 1.7254 - val_accuracy: 0.2214
Epoch 2/20
34/34 [==============================] - 75s 2s/step - loss: 1.6682 - accuracy:
0.2199 - val_loss: 1.6343 - val_accuracy: 0.2214
Epoch 3/20
34/34 [==============================] - 74s 2s/step - loss: 1.6131 - accuracy:
0.2171 - val_loss: 1.6274 - val_accuracy: 0.2214
Epoch 4/20
34/34 [==============================] - 74s 2s/step - loss: 1.4661 - accuracy:
0.3015 - val_loss: 1.5836 - val_accuracy: 0.2664
Epoch 5/20
34/34 [==============================] - 74s 2s/step - loss: 1.2436 - accuracy:
0.4519 - val_loss: 1.2546 - val_accuracy: 0.4296
Epoch 6/20
34/34 [==============================] - 74s 2s/step - loss: 1.1654 - accuracy:
0.4965 - val_loss: 1.2183 - val_accuracy: 0.4822
Epoch 7/20
34/34 [==============================] - 74s 2s/step - loss: 1.0859 - accuracy:
0.5527 - val_loss: 1.0450 - val_accuracy: 0.5591
Epoch 8/20
34/34 [==============================] - 74s 2s/step - loss: 1.0688 - accuracy:
0.5626 - val_loss: 0.9960 - val_accuracy: 0.5779
Epoch 9/20
34/34 [==============================] - 74s 2s/step - loss: 1.0382 - accuracy:
0.5734 - val_loss: 0.9831 - val_accuracy: 0.5929
Epoch 10/20
34/34 [==============================] - 74s 2s/step - loss: 0.9951 - accuracy:
```

```
0.6053 - val_loss: 1.0369 - val_accuracy: 0.5647
Epoch 11/20
34/34 [==============================] - 74s 2s/step - loss: 0.9898 - accuracy:
0.6057 - val_loss: 1.0076 - val_accuracy: 0.6098
Epoch 12/20
34/34 [==============================] - 74s 2s/step - loss: 0.9969 - accuracy:
0.6067 - val_loss: 1.0712 - val_accuracy: 0.6004
Epoch 13/20
34/34 [==============================] - 74s 2s/step - loss: 0.9382 - accuracy:
0.6418 - val_loss: 1.1118 - val_accuracy: 0.5854
Epoch 14/20
34/34 [==============================] - 75s 2s/step - loss: 0.8850 - accuracy:
0.6639 - val_loss: 0.8923 - val_accuracy: 0.6867
Epoch 15/20
34/34 [==============================] - 74s 2s/step - loss: 0.8577 - accuracy:
0.6765 - val_loss: 0.9122 - val_accuracy: 0.6773
Epoch 16/20
34/34 [==============================] - 74s 2s/step - loss: 0.8721 - accuracy:
0.6737 - val_loss: 0.9875 - val_accuracy: 0.6492
Epoch 17/20
34/34 [==============================] - 74s 2s/step - loss: 0.8460 - accuracy:
0.6728 - val_loss: 0.9034 - val_accuracy: 0.6604
Epoch 18/20
34/34 [==============================] - 75s 2s/step - loss: 0.8078 - accuracy:
0.7023 - val_loss: 0.8607 - val_accuracy: 0.6717
Epoch 19/20
34/34 [==============================] - 75s 2s/step - loss: 0.8248 - accuracy:
0.6990 - val_loss: 1.2641 - val_accuracy: 0.6173
Epoch 20/20
34/34 [==============================] - 75s 2s/step - loss: 0.7879 - accuracy:
0.7098 - val_loss: 0.9397 - val_accuracy: 0.6886
```






```
40/40 [==============================] - 14s 337ms/step - loss: 0.9656 -
accuracy: 0.6655
CURRENTLY TRAINING THE MODEL WITH THE FOLLOWING PARAMS:
{'l2_param': 0.0001}
Epoch 1/20
```

1.3 Training the Model

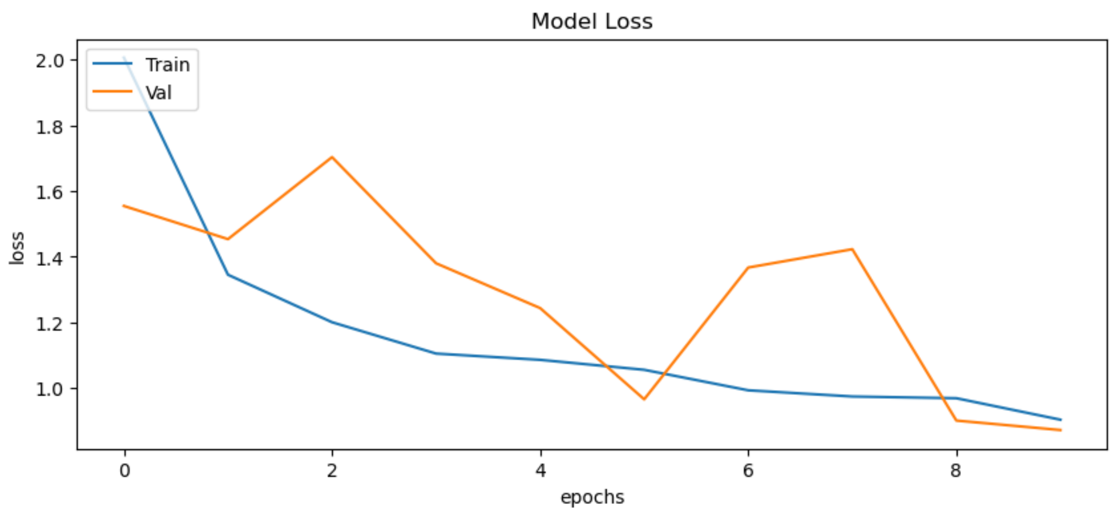
[ ]:
```
model = create_model()
model.summary()
```

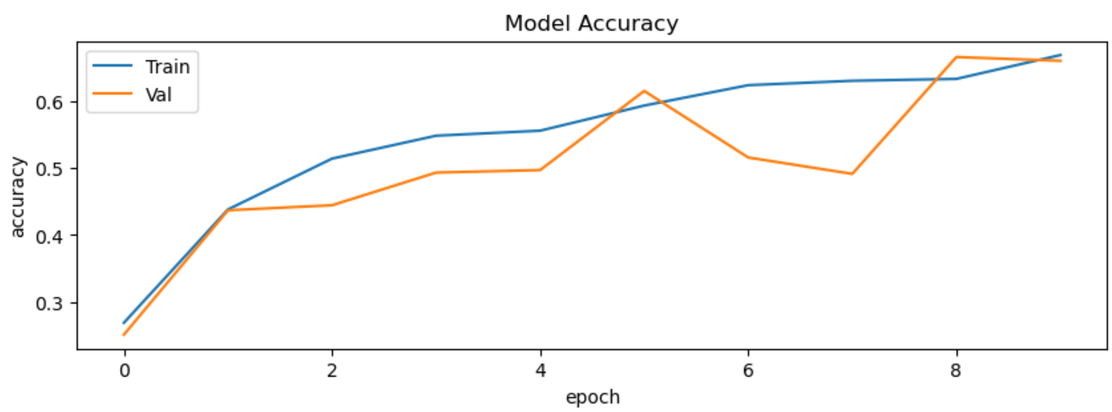
[ ]:
```
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

```
Epoch 1/10
34/34 [==============================] - 75s 2s/step - loss: 2.0064 - accuracy:
0.2691 - val_loss: 1.5546 - val_accuracy: 0.2514
Epoch 2/10
34/34 [==============================] - 75s 2s/step - loss: 1.3451 - accuracy:
0.4383 - val_loss: 1.4536 - val_accuracy: 0.4371
Epoch 3/10
34/34 [==============================] - 75s 2s/step - loss: 1.2005 - accuracy:
0.5143 - val_loss: 1.7033 - val_accuracy: 0.4447
Epoch 4/10
34/34 [==============================] - 128s 4s/step - loss: 1.1050 - accuracy:
0.5485 - val_loss: 1.3800 - val_accuracy: 0.4934
Epoch 5/10
34/34 [==============================] - 75s 2s/step - loss: 1.0859 - accuracy:
0.5560 - val_loss: 1.2435 - val_accuracy: 0.4972
Epoch 6/10
34/34 [==============================] - 75s 2s/step - loss: 1.0556 - accuracy:
0.5935 - val_loss: 0.9657 - val_accuracy: 0.6154
```

```
Epoch 7/10
34/34 [==============================] - 75s 2s/step - loss: 0.9931 - accuracy:
0.6240 - val_loss: 1.3670 - val_accuracy: 0.5159
Epoch 8/10
34/34 [==============================] - 75s 2s/step - loss: 0.9743 - accuracy:
0.6306 - val_loss: 1.4229 - val_accuracy: 0.4916
Epoch 9/10
34/34 [==============================] - 75s 2s/step - loss: 0.9690 - accuracy:
0.6334 - val_loss: 0.9010 - val_accuracy: 0.6660
Epoch 10/10
34/34 [==============================] - 75s 2s/step - loss: 0.9036 - accuracy:
0.6690 - val_loss: 0.8721 - val_accuracy: 0.6604
```

[ ]:
```
%run rueegg_wissiak_model_visualization.ipynb
```





[ ]:
```
model.evaluate(test_ds)
```

```
40/40 [==============================] - 13s 334ms/step - loss: 0.9018 -
accuracy: 0.6615
```

[ ]:
```
[0.9018300771713257, 0.6614645719528198]
```