Callealta Andres
A01763701@tec.mx

Birds species recognition with AI

"A diverse collection of six unique bird species for image classification" -Rahma Sleam

Problem selection; why this project ?

- Imagine a blind person that wants to go to an ornithological park, to listen to the birds, yes he can listen but does not know which species are there. The point of this "project" is to implement a Convolutional Neural Network model and to fine tuning it. The model will be able to recognize bird's species by taking pictures as input.

Information about the dataset:

- The dataset chosen is composed of different folders, ordered by bird species and weighs 15.97MB, a pretty small set for a six class classification model, but that will be the challenge.
- Images can be found here: https://www.kaggle.com/datasets/rahmasleam/bird-speciees-dataset

Each folder:

1. American Goldfinch, 143 images
2. Barn Owl, 129 images
3. Carmine bee-eater, 131 images
4. Downy wood-pecker, 137 images
5. Emperor Penguin, 139 images
6. Flamingo, 132 images

Distribution of the set: For this project, as the set is pretty small, I decided to separate it in 70% for training (529 images), 15% for validation and test, 124 each. I decided to do it like that, without privilege training set, because otherwise the test results might exhibit too much variance, making them unreliable for assessing the model's true performance. When the test set is too small, minor variations in its composition can lead to significant differences in performance metrics, giving a misleading impression of the model's generalization capabilities.

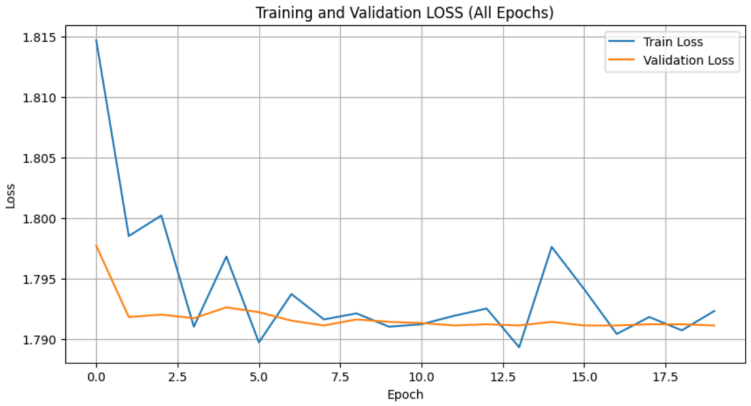
To begin with our project, we will begin with implementing two different models, then check the results and choose the one we will be working on:

- EfficientNetB0 is the baseline model of the EfficientNet family, a set of state-of-the-art convolutional neural networks designed for image classification tasks. Introduced by Google in 2019, EfficientNetB0 achieves a remarkable balance between accuracy and computational efficiency. Achieves

Callealta Andres
A01763701@tec.mx

77.1% top-1 accuracy on the ImageNet dataset with fewer parameters and resources compared to larger models.

- 5.3 millions of parameters
- 82 layers

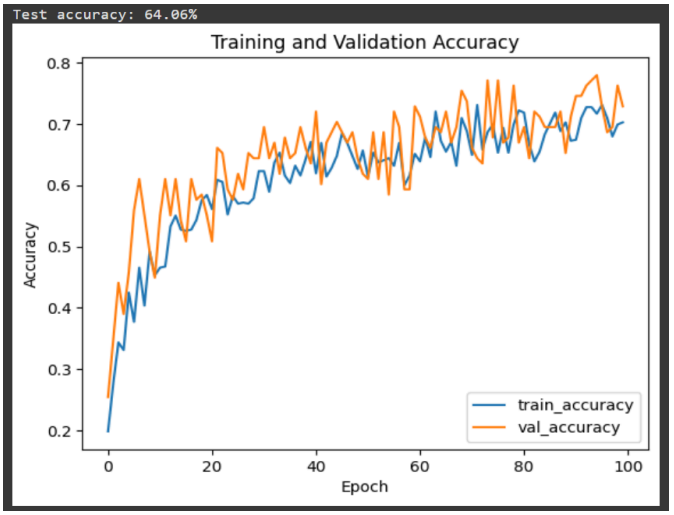


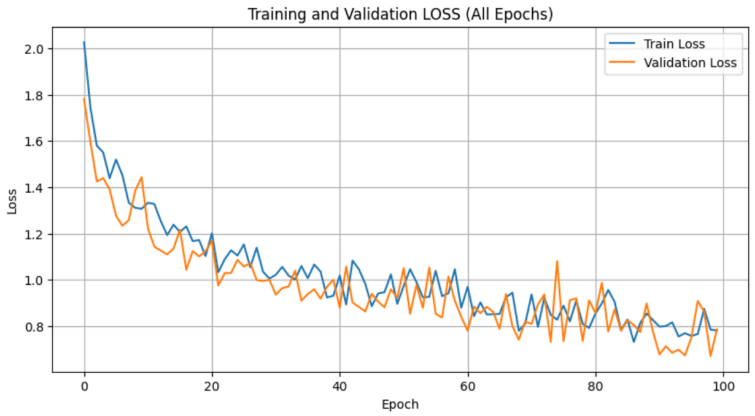
Model A: EfficientNetB0

20 epochs; learning rate: 0.001; loss function: 'categorical_crossentropy'

- ResNet-50 is a deep convolutional neural network (CNN) introduced by Microsoft in the landmark paper "Deep Residual Learning for Image Recognition" (2015). It is part of the ResNet (Residual Network) family, designed to address the vanishing gradient problem and enable training of very deep networks.
  - 50 layers
  - 25.6 millions parameters

Callealta Andres
A01763701@tec.mx





Model B: ResNet50
100 epochs; learning rate: 0.001, loss: 'categorical_crossentropy'

- Choice based on the results: I decided to continue with the ResNet 50 model because it has better results, for the same number of epochs, 20, model A has an accuracy of only 17%, much worse than Model B at the 20th epoch: ~50%. Added to that, we see a bigger variance, comparing train and validation results.

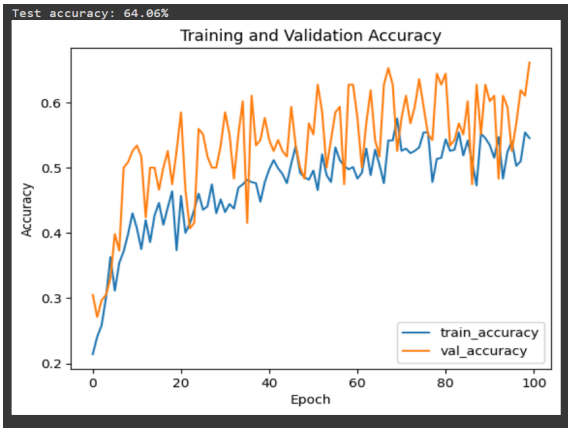
We clearly see a difference between accuracy on training results of ~72% and accuracy on tests: ~65.6%. This is a sign of overfitting. Due to our hyperparameter first selection and the small size of the dataset, our model is memorizing too much and not getting the bird's patterns. Objectively the difference in accuracy between the training and test sets is not extremely large, considering the small size of the dataset, it indicates significant variance in the validation and test results. This variability makes it difficult to reliably assess the model's true performance.

For that reason we will use a regularization method, considering also than our dataset is small, we will begin with data augmentation, below we ca find the transformations we do, there are basic operations, like changing the brightness, zoom or shifts:

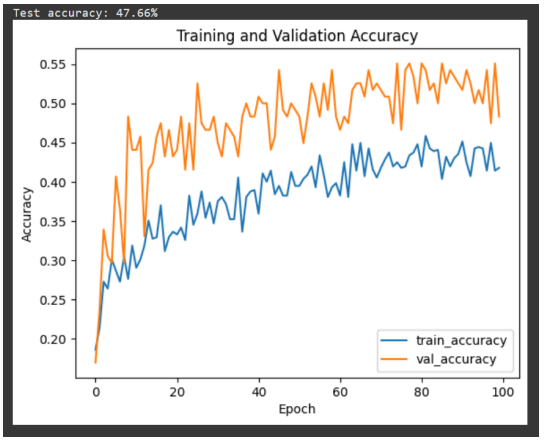
Callealta Andres
A01763701@tec.mx

```
train_datagen = ImageDataGenerator(
    rescale=1.0/255, :Normalizes pixel values to bring them into a range of 0 to 1
    rotation_range=15, :Rotates images randomly within a range of -15 to +15 degrees.
    width_shift_range=0.1, :Randomly shift images horizontally
    height_shift_range=0.1, :Randomly shift images vertically
    shear_range=0.1, :Applies shear distortion to images, distorts the image by skewing its contents horizontally or vertically up to 10%.
    zoom_range=0.1, :Randomly zooms in or out (+/- 10%)
    horizontal_flip=True :Randomly flip images horizontally
)
```

With that, I will try many different setups of hyperparameters:



Model D
100 epochs, learning rate: 0.001, batch_size: 32
Noise also added to the images



Model E
100 epochs; learning rate: 0.001; batch_size 32
Dropout(0.5)

After trying many regularization methods, like dropping the batch size, adding more data augmentation, changing the optimizer to AdamW, dropout, the learning rate, or even changing the cost and activation function or adding more epochs, the model was getting less and less robust. Finally I tried the more basic one possible, without data augmentation and it had the best results. Sometimes maybe because of the set size, the model performs better with less complexity. So finally, from a model with 60% on the test accuracy, we passed thru 50% and finally 68% with the more basic model.

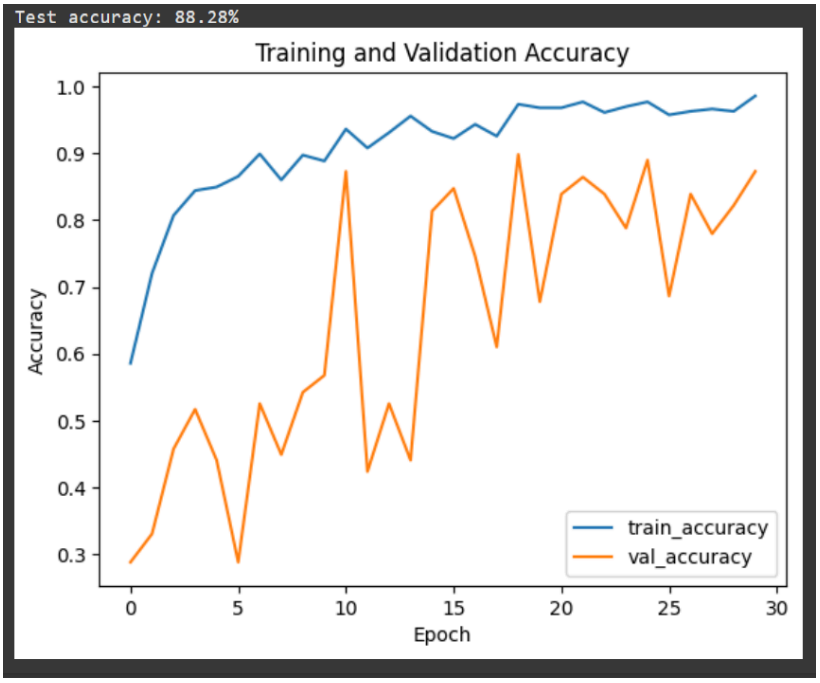
Callealta Andres
A01763701@tec.mx

Model complexity versus dataset size:

Deeper models excel at capturing features across various abstraction levels, but they also require significantly larger datasets to perform effectively. In this project, the limited size of our dataset (600 images) proved challenging for complex architectures. EfficientNetB0 (82 layers) yielded poor results because its complexity far exceeded the capacity of our dataset to provide sufficient training examples. While ResNet50 (50 layers) performed better, likely due to its more balanced design, the results still highlighted overfitting and suboptimal generalization, and it was really hard to improve it. EfficientNetB0's slight advantage can be attributed to its pre-trained weights, which leverage knowledge from a large-scale dataset (1 million images across 1,000 categories). However, given the persistent overfitting and the limited generalization capabilities observed, we opted to reduce model complexity further by using ResNet18 (18 layers). This smaller model aligns better with the constraints of our dataset, offering a more practical trade-off between complexity and performance.

For those reason, I will use a small model: ResNet 18, instead of a more complex like ResNet 50,

-

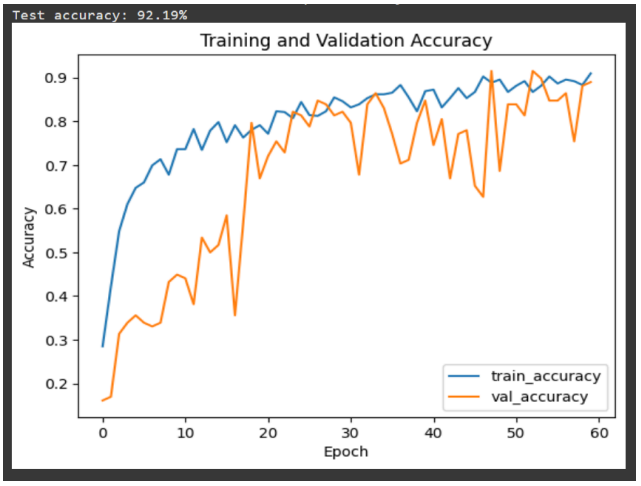


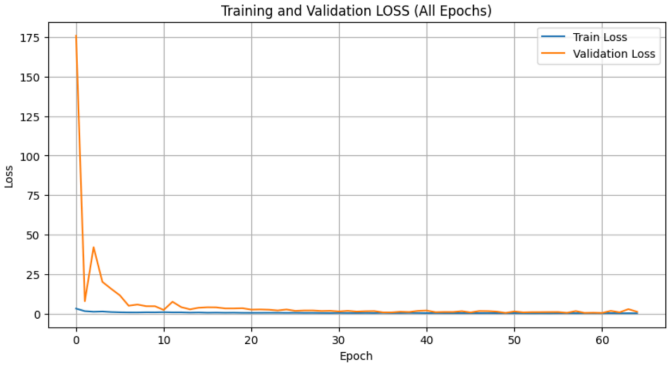
Model F ResNet 18:

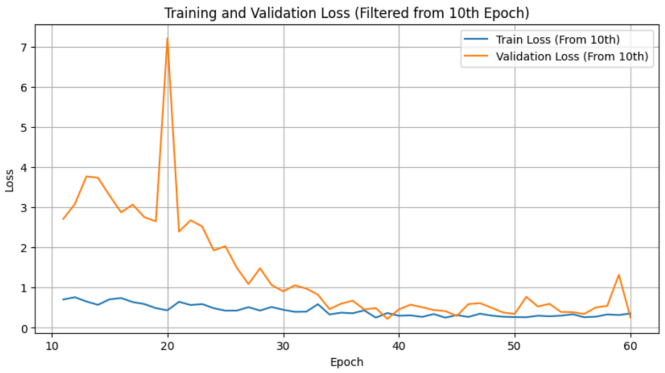
30 epochs, learning rate: 0.001, batch_size: 16

Finally we have a decent result, 88.28% in the accuracy, in only 30 epochs. We see a large difference between train_accuracy curve ~98% and val_accuracy ~87%, this can be a little sign of overfitting, the model is memorizing and not getting enough patterns. For that reason, I will first add dropout (0.3), a larger batch size and epochs, I will put it on 32. A larger batch size can help stabilize the learning process by providing more representative gradients

during each update. With a batch size of 32, the model will process more samples at once, reducing noise in the gradient estimation and making the training process smoother. This can lead to better generalization by preventing the model from overfitting to small, noisy variations in the data. Additionally, combining a larger batch size with dropout (0.3) will further regularize the model, encouraging it to focus on meaningful patterns rather than memorizing the training data. We will also change the learning rate, until here it was 0.001 and now we change it to 0.0001. There a the results of the model fine tuned:







Model G, ResNet 18

60 epochs, learning rate: 0.001, batch_size:32, Dropout(0.3)

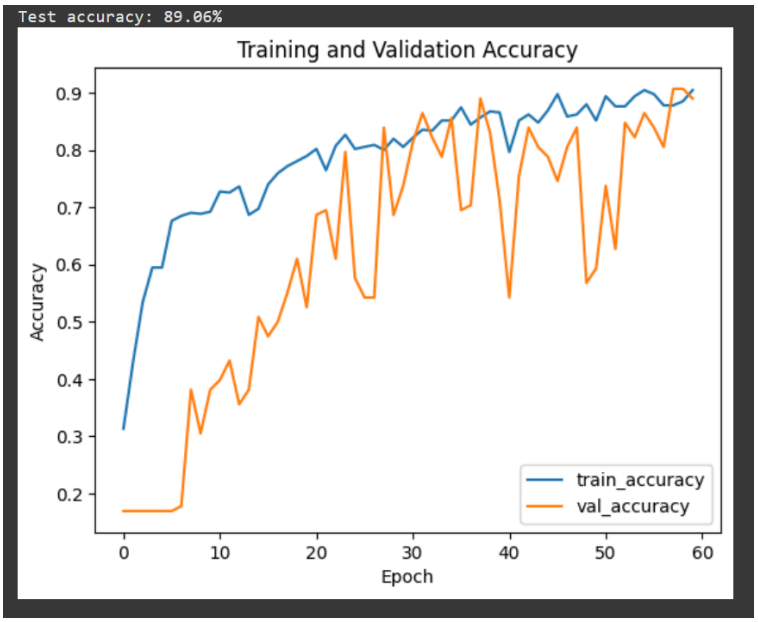
Callealta Andres
A01763701@tec.mx

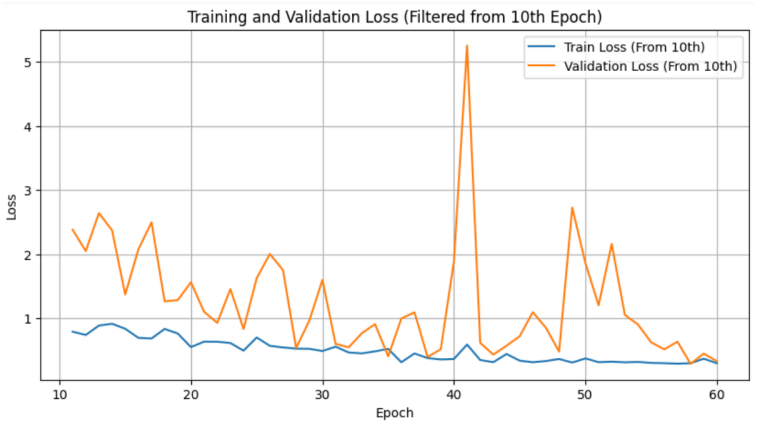
We see a notable variance in the curves, it is relatively normal due to our small size of the dataset. We see the last changes did work well on the model's performance, we went from a test accuracy of 88.25% to 92.19%. The model is globally sure when it is predicting a flamingo or an emperor penguin, near to 100% confident, but about 50-75% sure when it predicts other species.

Stochastic Depth (one last try):

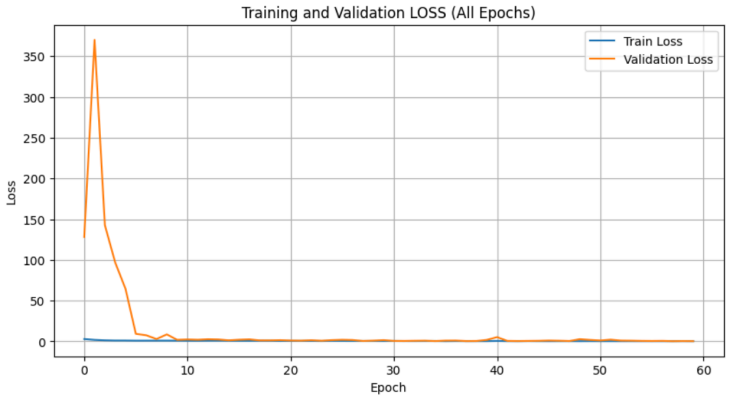
So far, we have explored and tested many regularization techniques, but not this one yet: Stochastic Depth. This method, described as belonging to the family of noise-based regularization techniques [source: Regularization in ResNet with Stochastic Depth (Hayou S. & Fadhel A.)], is specifically designed for ResNet models, which use residual blocks .

Unlike dropout, which involves removing certain neurons or weights each iteration, Stochastic Depth (SD) works by randomly disabling entire layers during training. Only the remaining layers are used to form a subnetwork whose weights are updated in each iteration. This particular mechanism makes SD a technique suited exclusively to residual neural networks (like ResNets).





Callealta Andres
A01763701@tec.mx



Model H, ResNet 18

60 epochs, learning rate: 0.001, batch_size:32, Dropout(0.3)

& Stochastic Depth

Explaining the results and conclusion:

Unfortunately, Stochastic Depth, removing randomly an entire layer, did not work well. It is strongly probable that removing a full layer can help on a more complex model, like ResNet50. Removing an entire layer in a shallower network like ResNet18 is likely too aggressive, leading to significant information loss and disrupted feature hierarchies (in the residuals blocks). This approach might be more effective in deeper, more complex models where the impact of removing a layer is less pronounced, and where the vanishing gradients are a real topic. This last point occurs in complex models when the gradients of the first layers become too small, here with 18 layers we probably do not have this problem. For shallower networks, a more nuanced regularization technique or a less aggressive form of Stochastic Depth might be more appropriate.

Our final model is the model G, ResNet18, with 60 epochs, a learning rate: 0.001, a batch_size: 32 and a dropout 0.3.

Callealta Andres
A01763701@tec.mx

Overview on the model's architecture, explained in comments:

```
def ResNet18(input_shape=(224, 224, 3), num_classes=6):
    ''' Input Layer
        7x7 convolution, 64 filters, stride=2
        Batch Normalization
        ReLU
        MaxPooling 3x3, stride=2 '''
    inputs = Input(shape=input_shape)
    x = Conv2D(64, kernel_size=7, strides=2, padding="same")(inputs
    )
    x = BatchNormalization()(x)
    x = ReLU()(x)
    x = MaxPooling2D(pool_size=3, strides=2, padding="same")(x)



    ''' Bloc 1
        Two residual blocks with 64 filters
        Convolution size: 3x3, stride=1 for both convolutions in each
    block
        Block output: same dimension as the input (56x56 if the input
    is 224x224)'''
    x = residual_block(x, 64)
    x = residual_block(x, 64)
    x = Dropout(0.3)(x)

    ''' Block 2
        Two residual blocks with 128 filters
        Convolution size: 3x3, stride=2 for the first convolution
        of the first block to reduce the resolution to 28x28, then
    stride=1'''
    x = residual_block(x, 128, stride=2)
    x = residual_block(x, 128)
    x = Dropout(0.3)(x)

    ''' Block 3
        Two residual blocks with 256 filters
        Convolution size: 3x3, stride=2 for the first convolution
        of the first block to reduce the resolution to 14x14, then
    stride=1'''
    x = residual_block(x, 256, stride=2)
    x = residual_block(x, 256)
    x = Dropout(0.3)(x)

    ''' Bloc 4
        Two residual blocks with 512 filters
        Convolution size: 3x3, stride=2 for the first
        convolution of the first block to reduce the resolution to 7x7
    '''
    x = residual_block(x, 512, stride=2)
    x = residual_block(x, 512)

    ''' Output Layer
        Global Average Pooling: Reduces each 7x7 trait card to a
    single value
        Dense (Fully Connected) Layer:
        Number of neurons = number of classes (for example, 1000 for
    ImageNet)
        Softmax activation for final classification'''
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.5)(x)
    outputs = Dense(num_classes, activation="softmax")(x)
    model = Model(inputs, outputs)
    return model
```