Object Classification/Recognition using CNN Networks and Transfer-learning with EfficientNet-B0

Presented by:
Akanksha Rawat
Karishma Kuria
Nisha Mohan Devadiga

Team members

1. Akanksha Rawat
2. Karishma Kuria
3. Nisha Mohan Devadiga

Introduction

- Object detection and image recognition is complex task for machines due to factors such as:
  - Amount of light in the image.
  - Angle.
  - Position of the object.
- Convolutional Neural Network (CNN) has shown good performance in the field of image classification and Object detection.
- CNN has a great ability of hierarchical feature learning.
- The aim of this project is to develop a CNN model using transfer learning that can accurately identify and categorize colored photographs of objects into one of the 100 available classes.
- We have used Canadian Institute For Advanced Research (CIFAR-100) dataset which is a labeled dataset containing 80 million tiny images.
  - Dataset comprises 60,000 colored images of 32 by 32 pixels
  - 100 classes.
  - Divided into 20 super classes (50,000 training and 10,000 test)

Related Work

- Data Augmentation
- Transfer learning
- Pretrained Model - ***EfficientNet***

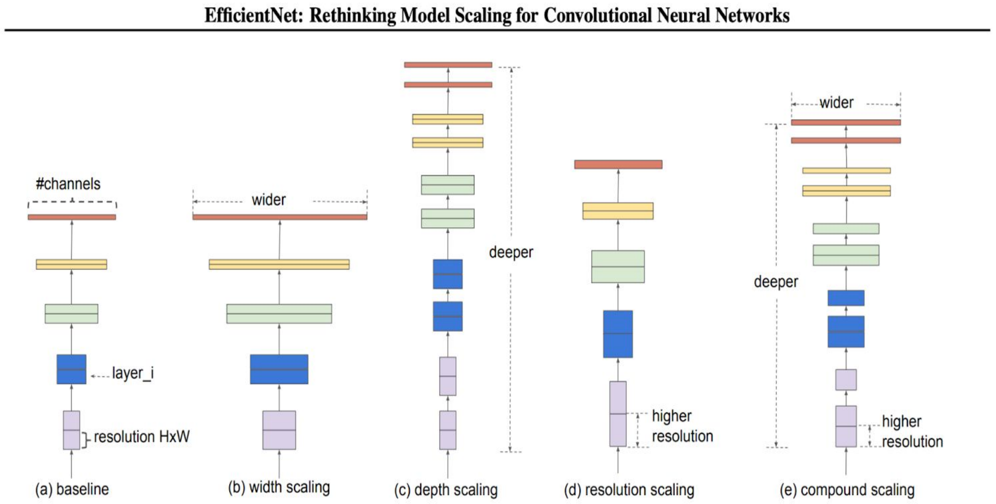


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

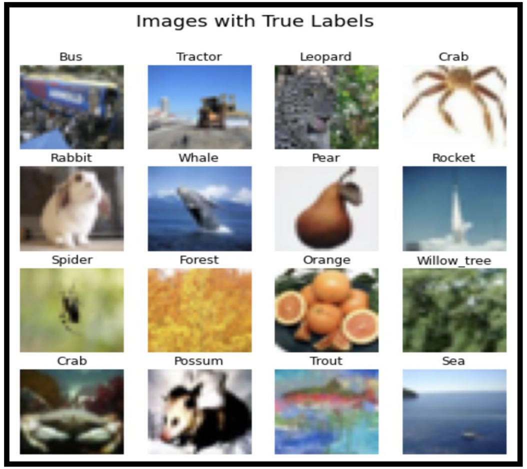
Image Source - TowardsDataScience [link]

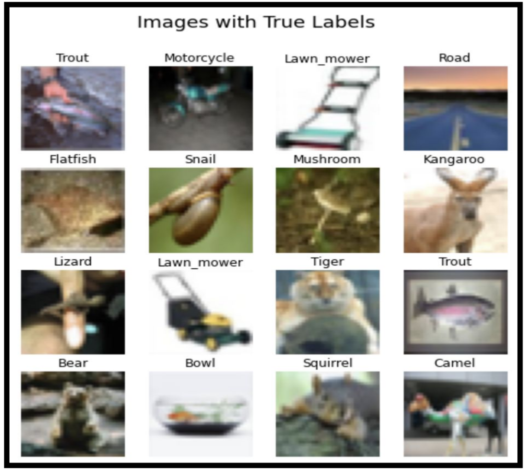
Business Understanding

- A prototype for an interactive system that can classify objects from input image frames.
- The use of this model in object analysis tools to identify and categorize colored images is a potential future use.
- The challenge of achieving a high accuracy score (higher than 59% as achieved with a 9-layer convolutional neural network built earlier) is the motivation behind using Transfer learning.
- The visual quality of this dataset is very intriguing.
- In order to teach the computer to correctly recognize and classify the photographs more precisely than it did previously, transfer learning has been used.

Data Understanding

- CIFAR-100 has 100 classes but just 600 images in each class (500 for training and 100 for testing).
- Each of the images in the dataset is of 32 × 32 pixels which makes recognition a challenging task for machines.
- Memory is the biggest obstacle to creating a deep neural network for CIFAR-100 that has millions of parameters.
- We have used a balanced dataset and used data augmentation to expand the training dataset using techniques like Image Shifting, Image Flipping, Image Zooming.





Data Preparation

- We have used python version of the dataset.
- For the purpose of serialization or deserialization of these objects in Python, the Pickle module has been utilized.
- Several functions have been added to create dataframe with features such as coarse label (superclass) and label(class).
- EDA and data cleaning is done.
- Data batches are generated which contains both labels and images.
- Data normalization is done before feeding the data to various object classification models.

Introduction - Weights & Biases(W&B)

- Platform for enabling a collaborative **MLOps** culture.
- Designed to support and automate key steps in the MLOps life cycle - experiment tracking, dataset versioning and model management.
- Key pillars of MLOps -
    - ability to carry out various experiments
    - tracking different configuration that affects the model metrics

Image Source -https://github.com/wandb/wandb

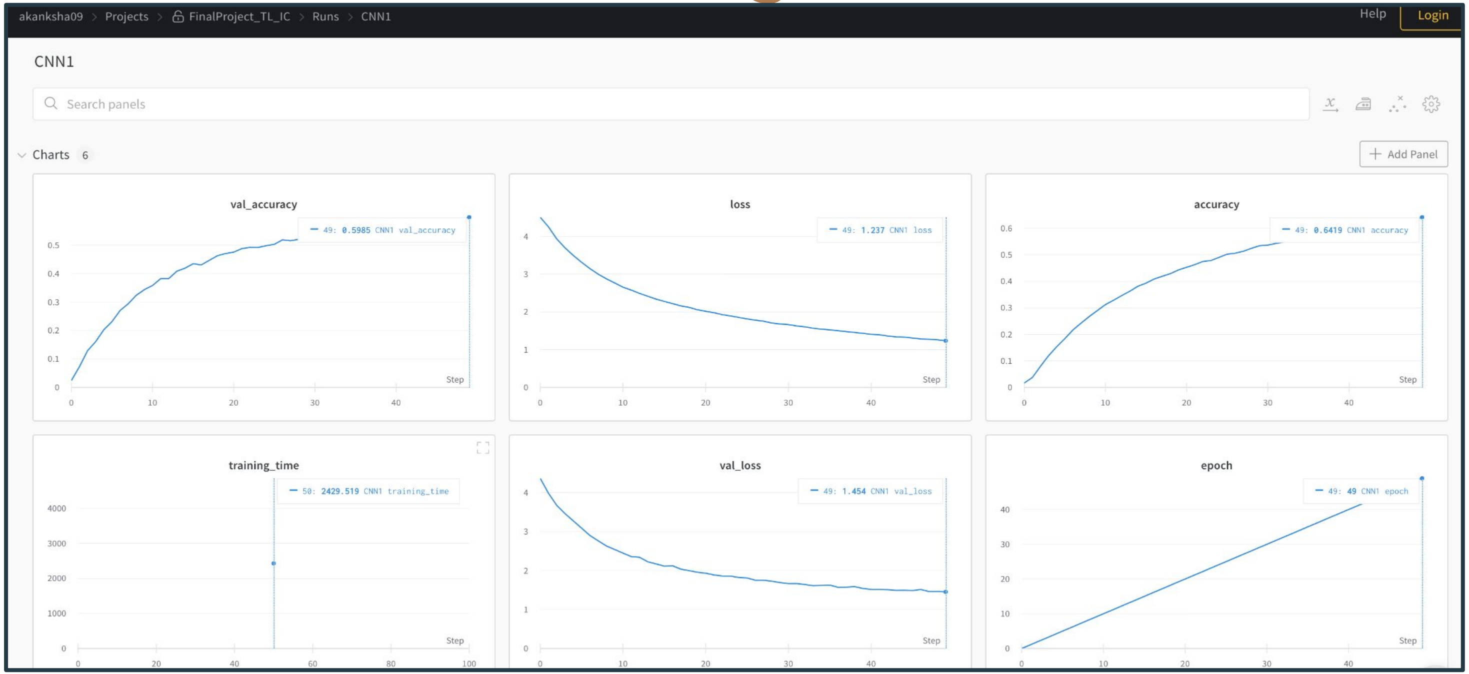
Setup Weights & Biases account and create project

- import wandb
  wandb.login()

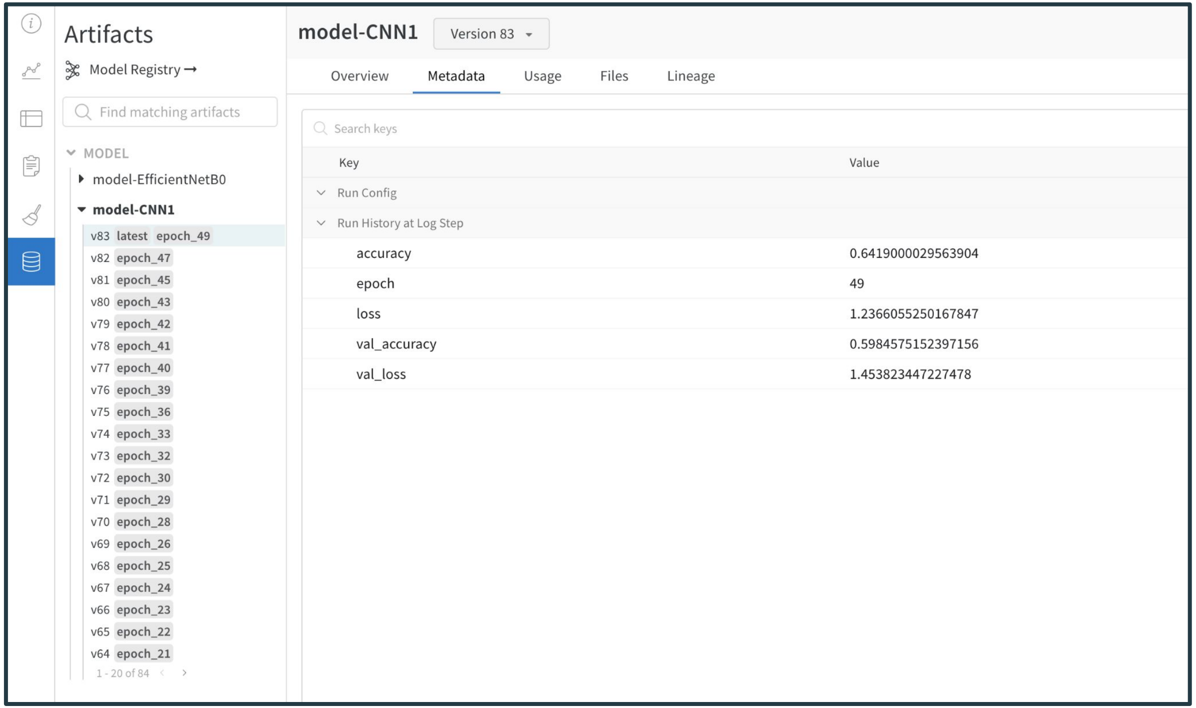
Initialization:

- wandb.init(project="FinalProject_TL_IC", entity="akanksha09")

Weights & Biases, wandb.ai for model/artifacts tracking



Weights & Biases, wandb.ai for model/artifacts tracking



Modeling

1. **Fully Connected CNN Model**

In the first section, we created an architecture for Fully Connected CNN and a multi-layered Tensorflow framework using the Keras high level API.

- The output contains 100 values which shows 20 categories.
- Number of filters tends to increase with depth of the model when more representational capacity is required in the model.
- Size of filters is mostly evenly distributed.
- We have used 3 stacks of layer combinations - Conv2D , Conv2D, MaxPool2D and Dropout layers.
- Each stack has two Conv2D layers with the same padding , ReLu activation, followed by MaxPool2D with pool size of 2, strides of 2.
- The last layer is the softmax activation.

Modeling

2. EfficientNet Model

- The standard CNN model is systematically studied for scaling and identifies that carefully balancing network depth, width, and resolution
- We use stratified shuffle split to preserve the percentage of samples in each of the 100 classes.
- Overfitting was prevented by randomly sampling the outputs of the dropout-related layers.
- The Adam optimization algorithm has been utilized in the model.
- Categorical cross entropy loss was utilized because the dataset required multiple class classification.
- The model has employed early stopping and reduced learning rate on plateau strategies to track validation loss.
- The model was trained using an 8-batch size across 15 epochs.

EfficientNet Model Summary

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnet-b0 (Functional  (None, 7, 7, 1280)       4049564
 )

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 100)               128100

=================================================================
Total params: 4,177,664
Trainable params: 4,135,648
Non-trainable params: 42,016
_________________________________________________________________
```

Model Compilation

We have compiled our model with parameters - Loss and Accuracy.

```
optimizer = Adam(lr=0.0001)
#early stopping to monitor the validation loss and avoid overfitting
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=10, restore_best_weights=True)

#reducing learning rate on plateau
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience= 5,
factor= 0.5, min_lr= 1e-6, verbose=1)

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

Model Training

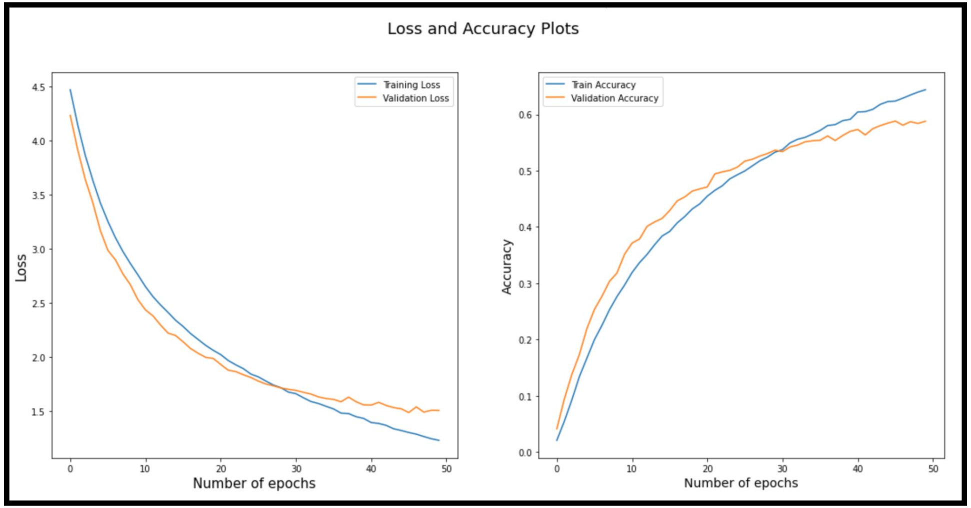
- We have used ReduceLROnPlateau technique for reducing the learning rate when a metric stops getting better.
- To train our model, we have used the ‘fit()’ method on our model with the following parameters: training data , steps_per_epoch, callbacks, validation data, validation steps, and the number of epochs.
- We have used 15 epochs for training.

```
model_history = model.fit_generator(train_data_generator,
                                    validation_data=valid_data_generator,
                                    callbacks=[early_stop, rlrop],
                                    verbose=1,epochs=epochs)
```

Experiment - 1

Fully Connected CNN Model

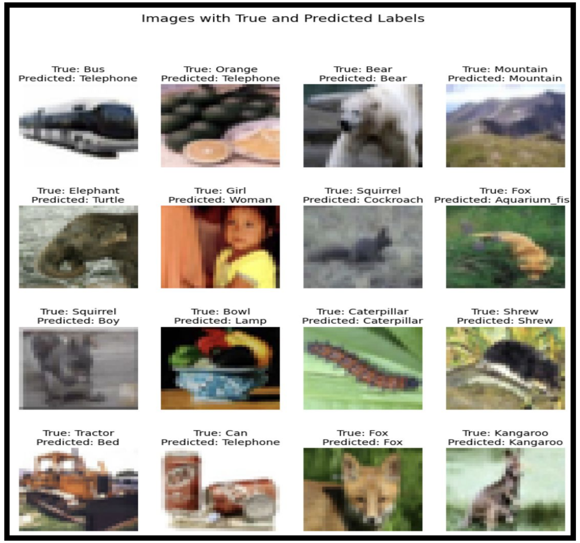
- As the number of epochs increases, both training and validation loss gradually decreases.
- After epoch = 28, validation loss is higher than the training loss. → means that the model is overfitting to the train dataset and failing to generalize to the validation dataset.
- Similarly until epoch = 28, Validation accuracy is higher than training accuracy which indicates that the model has generalized fine.
- The model completes with the validation accuracy of 59.96 % and test accuracy of 59.82 % and notes the validation loss of 1.44 and test loss of 1.43.



Experiment - 1

Fully Connected CNN Model

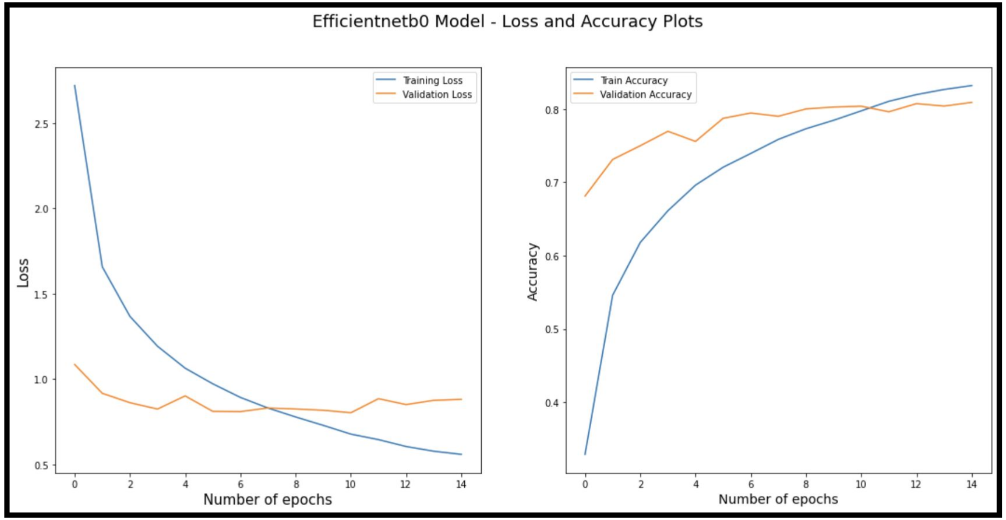
The result for true and predicted image after model validation can be seen here →



Experiment - 2

EfficientNet

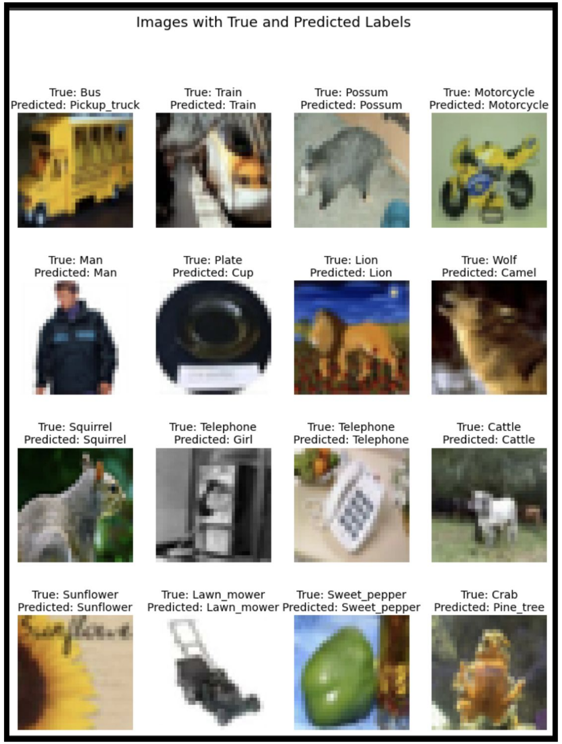
- As the number of epochs increases, both training loss gradually decreases.
- Validation loss oscillates around 0.9 to 1. It can be noted that after epoch = 7, validation loss becomes higher than the training loss.
- Validation Accuracy of 80.89 % and Test Accuracy of 80.55 % .



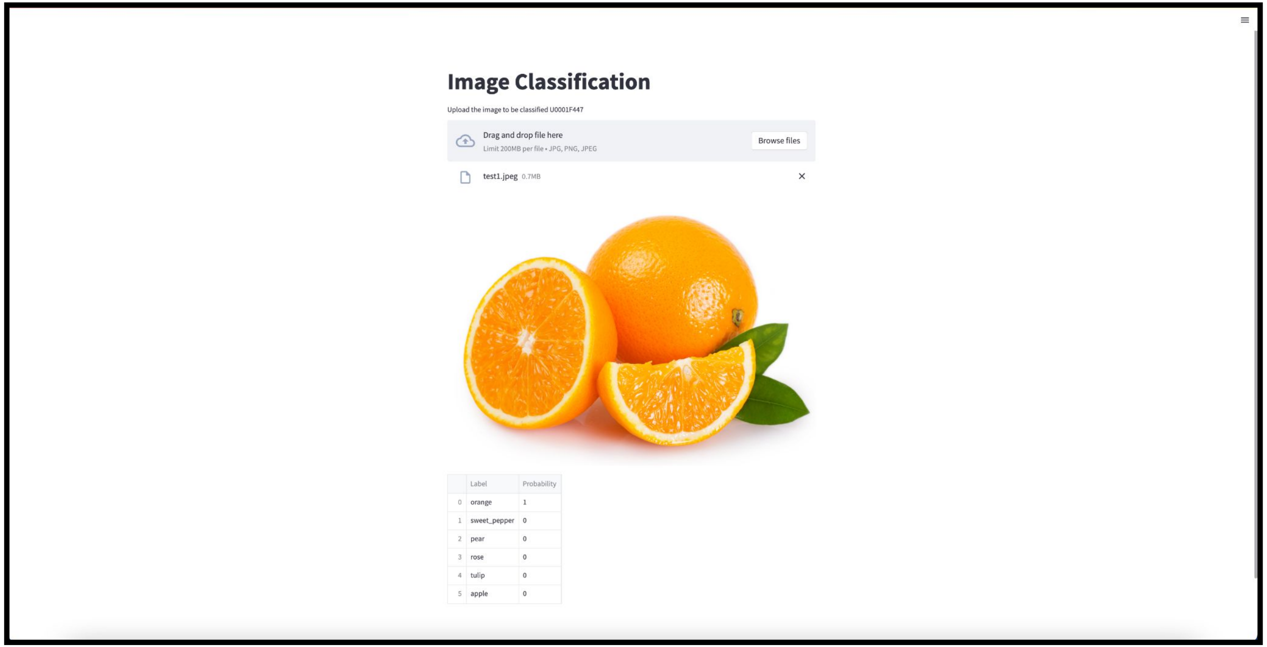
Experiment - 2

EfficientNet-B0

The result for true and predicted image after model validation can be seen here →



Application / Model Prediction using Streamlit



DEMO

Application Link

References

- https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4
- https://www.irjet.net/archives/V7/i11/IRJET-V7I11204.pdf
- https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8
- https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/
- https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/

Deployment Reference

- Building a Playground with Streamlit - https://hackernoon.com/how-to-use-streamlit-and-python-to-build-a-data-science-app
- Heroku deployment without the app being at the repo root (in a subfolder)

Article:

- https://coderwall.com/p/ssxp5q/heroku-deployment-without-the-app-being-at-the-repo-root-in-a-subfolder
- https://github.com/timanovsky/subdir-heroku-buildpack
- Heroku how to switch deployment from github to heroku-git with app changes in github - https://help.heroku.com/CKVOUPSY/how-to-switch-deployment-method-from-github-to-heroku-git-with-all-the-changes-app-code-available-in-a-github-repo