CSE 144, Winter 2024: Final Project Report

Sneha De (snde@ucsc.edu), Karthi Sankar (kasankar@ucsc.edu)

Model Architecture

Our model uses the EfficientNetV2B3 architecture hosted by Keras.

1. Initially, the necessary libraries are loaded, and many standard values are set, such as the seed for augmentation and image/batch size, among others.
2. Before passing in the inputs into the base model, various data augmentations are randomly done to the training data such as rotations, flips, and changes in contrast.
3. EfficientNet models include a preprocessing layer, so the model is directly specified with the augmented images passed in as the input.
4. The last 10 layers of the base model are unfrozen to train upon our data, with the initial layers left intact.
5. After the base model, there is a dense layer of 100 neurons, a batch normalization layer, and a dropout layer with a probability of 0.3.
6. Lastly, a softmax dense layer of 100 neurons is applied to give the outputs.

Model Training

1. The constant for steps per epoch is defined as our batch size (32) times 10 for each of the 10 epochs.
2. The learning rate schedule is specified using ExpontentialDecay, and the optimizer used is Adam (both from the Keras API). As more steps are completed, the learning rate decays to approach the minimum loss more granularly.
3. An early stopping callback is set up, so if the validation loss doesn't improve for a certain number of epochs (in this case, the patience is 2 epochs), then training will stop to prevent overfitting.
4. An instance of the model is created and compiled using the chosen optimizer, loss function (sparse_categorical_crossentropy), and metric of accuracy.
5. The validation and training datasets are prefetched to help with performance.
6. Finally, the model is trained using the training data (set to repeat until all steps in the epoch are complete) and the validation data.

Model Summary

Model: "our_model"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 300, 300, 3)	0
augmentations (Sequential)	(None, 300, 300, 3)	0
efficientnetv2-b3 (Functional)	(None, 1536)	12,930,622
dense (Dense)	(None, 100)	153,700
batch_normalization (BatchNormalization)	(None, 100)	400
dropout (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 100)	10,100

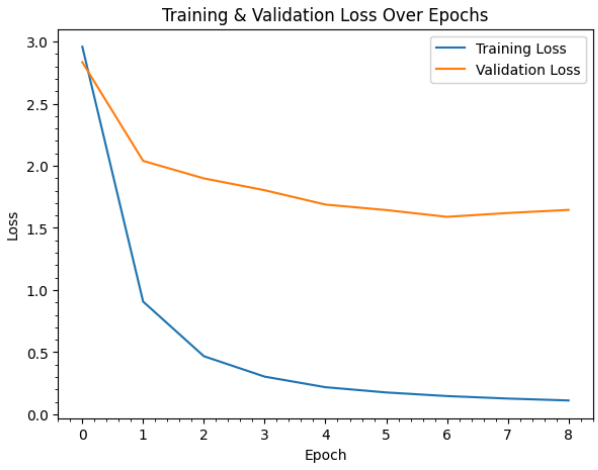
Total params: 13,094,822 (49.95 MB)

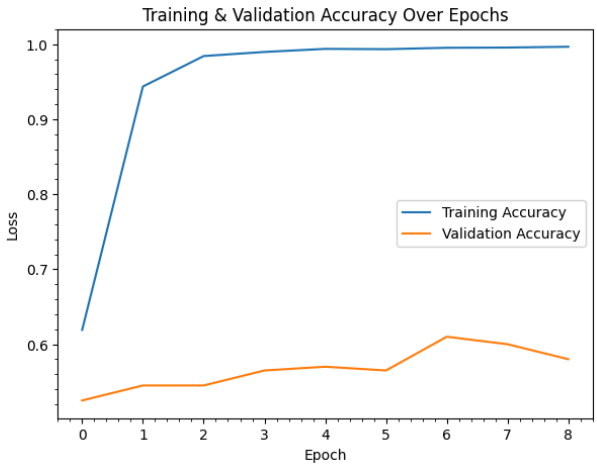
Trainable params: 928,960 (3.54 MB)

Non-trainable params: 12,165,862 (46.41 MB)

Results

Our model receives a public score in the range of [0.635, 0.655] on Kaggle depending on how much percent dropout we use, and the following is a chart of our typical training and validation loss and accuracy:





Environment

Our environment requires the following libraries:

- tensorflow==2.11.0
- keras
- kaggle
- numpy
- matplotlib (for graphics)

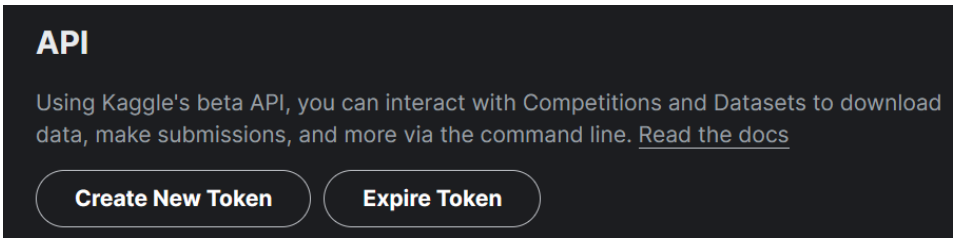
Other libraries used in the project are zipfile and csv, which are standard Python libraries.

Our notebook is known to run best on TensorFlow version 2.11.0 because this is the version our Google Cloud workbench provided a kernel for.

The beginning of our notebook ensures these dependencies are installed in the kernel:

```
!pip install -q tensorflow==2.11.0
!pip install -q --upgrade keras
!pip install -q kaggle numpy matplotlib
```

Usage

1. Retrieve a Kaggle API key in a `kaggle.json` file.
    a. Log into Kaggle
    b. Go to Account Settings
    c. Go to the API section and click "Create New Token"
        i. 

    d. A file called kaggle.json will be downloaded.
2. Place `kaggle.json` in the same directory as the notebook called "`cse144-final-project-notebook.ipynb`".
    a. It is located in our Google Drive folder.
3. Run all the cells in the notebook.
    a. The dataset will be downloaded automatically via the Kaggle API.
    b. The "`submission.csv`," which contains predicted classifications for the test data, and weights file will be generated from the model within the current directory.

Using our weights for transfer learning

The `model.loadweights("cse144-final-project-weights.weights.h5")` method can be used in other models using the EfficientNetV2B3 architecture.

Weights

A link to our model weights and the corresponding notebook can be found in our Google Drive Folder.

The file containing the weights is called "`cse144-final-project-weights.weights.h5`", and our notebook is called "`cse144-final-project-notebook.ipynb`"

Contributions

- **Sneha**
  - Set up notebook and streamlined pipeline for ease of use
    - Data loading, processing, augmentation, and exporting results
  - Experimented with data augmentation layers
  - Experimented with EfficientNet models for effectiveness
  - Experimented with various amount dense neurons, dropout rates, and rates of decay for the learning rate
- **Karthi**
  - Experimented with ResNet models and different layers for effectiveness
  - Tweaked hyperparameters for number of layers unfrozen in the base model and learning optimizers