Transfer Learning for CNN Image Classification and Image Manipulation

Veer, Ram, Ranjani

Abstract. In this paper, we are analyzing different variants of CNN models, specifically on the basis of how well they perform on the CIFAR-10 image set. In the process of this assignment, we initially begin with a baseline CNN model, which gave a sub-par accuracy of approx. 75%. To improve the accuracy, we then augmented the data and ran the PyTorch pre-trained ResNet model, which gave us an improved accuracy of 79%. We then tested the dataset on the PyTorch pre-trained DenseNet-121 model on our dataset, which seemed to perform much better, giving us an accuracy of 83%. We finally tested the dataset on a pre-trained model that is tailored to the CIFAR-10 dataset, which gave us a significant increase in accuracy, going from 83% in the PyTorch DenseNet model to almost 93% accuracy.

Key words. CIFAR-10, CNN, ResNet, DenseNet, Transfer Learning

1. Introduction.

The goal of this project is to classify images from the CIFAR-10 dataset using Convolutional Neural Networks (CNNs) and their variants. The CIFAR-10 dataset, consisting of 60,000 low-resolution (32x32) images across 10 distinct classes, presents a challenging image classification task due to its high variability. To tackle this, a baseline CNN model was implemented, followed by advanced techniques such as ResNet and DenseNet architectures, data augmentation, and dropout. Throughout the experimentation process, we continuously integrated successful techniques and parameters from previous models into subsequent iterations. Effective hyperparameters such as learning rates and epoch settings were carried over to enhance new models. The final models were evaluated using accuracy, precision, recall, F1-score, and loss curves to compare effectiveness and refine our approach to improving classification performance.

2. Related Work.

The CIFAR-10 classification problem is a well know classification problem thus we were able to review prior model creations, specifically by author "huynphan" on GitHub as well as other GitHub pages which helped us understand optimizers best used for this classification problem. However, there are many more implementations with the best current models achieving an accuracy of 99.5 percent (ViT-H/14 and DINOv2).

Another from of literature exploration was reading about transfer learning with LSTMs. Although the primary algorithm used in paper we read was not CNNs, we still were able to understand what transfer learning is and how it can help improve model creation for classification problems through this technique. This paper primarily focused on pre-training models with various feature inputs for binary stock classification, which is quite different to our problem, however the key idea being training models with large amounts of data and then using these models (best one or more) for fine-tuned specific problems can improve accuracy and efficiency of models.

Overall, we incorporate all three techniques from our literature review of related works. We made use of core ideas behind transfer learning by incorporation both torch models and the CIFAR10 models provided by "huynphan". We also experimented with various optimizers based on our optimizer choice reading, primarily focusing on Adam and Adamax optimizers.

Here we state our main result as **??**; the proof is deferred to section SM2ex$_s$*upplement.pdf*.

3. Problem set up (Modeling) and Data. The goal of this project is to classify images from the CIFAR-10 dataset using Convolutional Neural Networks (CNNs) and their variants. The CIFAR-10 dataset is widely used for image classification tasks and provides a robust benchmark for evaluating the performance of different neural network architectures.

3.1 Problem Setting. The CIFAR-10 dataset consists of **60,000 images** of size **32x32 pixels** in RGB color format, split into **50,000 training images** and **10,000 testing images**. The dataset is divided into **10 distinct classes**, representing common objects:

1. Airplane
2. Automobile
3. Bird
4. Cat
5. Deer
6. Dog
7. Frog
8. Horse
9. Ship
10. Truck

This classification task is challenging due to the low resolution of the images and the high variability within each class. The goal is to design and implement CNN-based models capable of accurately classifying these images into their respective categories.

5.2 Dataset Description. The CIFAR-10 dataset, provided by the **Canadian Institute for Advanced Research (CIFAR)**, is available through the torchvision and tensorflow libraries, making it easy to access and load. Each image is a 32x32x3 tensor representing **width, height, and three color channels (RGB)**. The dataset includes a balanced distribution of images across the 10 classes, with 5,000 images per class in the training set and 1,000 images per class in the test set.

5.3 Preprocessing Steps. To optimize model performance and reduce training time, the following preprocessing steps are applied:

1. **Normalization**: Each image's pixel values are scaled to the range [-1,1][-1, 1][-1,1] to stabilize the training process. The mean and standard deviation of the dataset are computed for normalization:
    - Mean: [0.4914, 0.4822, 0.4465]
    - Standard Deviation: [0.2023, 0.1994, 0.2010]
2. **One-Hot Encoding**: The class labels are converted to one-hot encoded vectors for compatibility with the neural network's output layer.

These preprocessing steps ensure that the dataset is well-prepared for training CNNs and provide additional variety in the training data, improving the model's robustness and generalization capabilities.

A **Convolutional Neural Network (CNN)** is a specialized type of neural network designed to process and classify visual data. Unlike traditional fully connected networks, CNNs leverage the spatial structure of images by applying **convolutional layers** to extract features such as edges, textures, and patterns.

A typical CNN consists of the following layers:

1. **Convolutional Layers**: Apply filters (kernels) that slide over the image to detect local patterns and features.
2. **Pooling Layers**: Reduce the dimensionality of the feature maps, typically through **max pooling** or **average pooling**, to make the network more computationally efficient.
3. **Fully Connected Layers**: Flatten the feature maps and connect them to a final output layer for classification.
4. **Activation Functions**: Non-linear functions (e.g., **ReLU**) applied to introduce non-linearity and improve learning capabilities.

During training, the CNN learns optimal filter values through **backpropagation** and **gradient descent** to minimize the classification error.

5.4 Problem Modeling. We frame the problem as a **multi-class classification** task. The network will output a probability distribution over the 10 classes, and the predicted class will be the one with the highest probability.

In this project, we implement the following models:

1. **Variants**:
    - **Data Augmentation and Drop out Model - Model 1**
    - **ResNet from Torch - Model 2**
    - **Densenet Model from Torch - Model 3**
    - **DenseNet from CIFAR-10 classifiers (benchmark model) - Model 4**
    - **DenseNet from CIFAR-10 classifiers (benchmark model) with adaptive learning rate - Model 5**

The models will be evaluated using metrics such as **accuracy, precision, recall, F1-score, confusion matrices and loss curves** to compare performance and effectiveness.

4. Methodology.

a. Model 1:

The dataset used is CIFAR-10 which is split into a train-test split, the split is determined by PyTorch native output of CIFAR-10 retrieval through the library call (the train data is shuffled but the test is not with a batch size of 128). The data is then augmented in three ways: a randomized horizontal flip for better generalization, random crop and padding of 4 pixels added on and finally normalized within a mean and standard deviation value passed in of [0.5,0.5,0.5] and [0.5,0.5,0.5] respectively which then normalized the data to a range of [-1,1].

The baseline model is a CNN (Convolutional Neural Network) that takes in the data preprocessed with the steps laid out in the above paragraph. The architecture of this baseline model is as follows: convolutional layers followed by fully connected layers. The input layer itself is natively handled by the PyTorch library. The convolution block is as follows: a convolution layer with 3 input channels and 32 output channels, kernel size of 3, padding of 1, the activation function used is ReLU, followed by a max pool layer with a kernel size of 2, stride of 2. The next layer is the second convolution layer with 32 input channels and 64 output channels, kernel size of 3, padding of 1 and again followed by a ReLU activation and a max pool layer with the same parameters. The outputs of the convolution layer is then

passed into a fully connected layer whose architecture is as follows: the data is flattened with a nn.Flatten( ) call. All this is followed by a linear layer with an input shape of 64*8*8 and output of 256, this layer uses ReLU activation and drop out rate of 0.5 and finally another linear layer that takes in an input of 256 and outputs of 10 (corresponding to the 10 classes).

The model explained above is used to train using the train data and once training is over the test data is passed into the model. The model evaluated on the train and test data with metrics: accuracy and loss scores as well accuracy and loss plots. Some additional details about the model are: a cross entropy loss, Adam optimizer and a learning rate of 0.001, 20 epochs of training iterations. We also make use of T4 GPU to improve model training time.

b. Model 2:

The data setup is similar to model 1 here but with a slight change to the normalization, rather than using 0.5 across the normalization pass for mean and standard deviation, benchmark values which are used for CIFAR-10 normalization are used instead. Where mean is now given a list: [0.4914, 0.4822, 0.4465] and standard deviation is given a list: [0.2023, 0.1994, 0.2010].

Model 2 is to experiment with torch's built in model for image classification. We made use of the resnet18. The architecture is as follows: our model is initialized with resnet18 and we add a linear layer at the end of the pre-trained model with a 10 class output shape. The results of both training and testing data are evaluated based on accuracy and loss plots. Other details regarding this model include: cross entropy loss for loss calculation, Adam optimizer with a learning rate of 0.001 and the epoch count of training is 5 epochs.

c. Model 3:

The data for this model is the exact same as model 2, there are no additional changes made. This model is an extension into exploring other torch models, specifically, we made use of the densest121 model. The model architecture is as follows, the denset121 model is added onto model 3, and a single final layer is added on top of this which is the linear layer that performs the class calculation. The model makes use of a cross entropy loss function and is optimized using the Adam optimizer, and a learning rate of 0.001. The training epoch count for this model is 10 epochs. The model is evaluated based on train, test accuracies as well as a loss plot for both train and test data. A GPU was used here as well.

d. Model 4:

Model 5 follows the same data setup as model 2. This model further highlights the use case of using pre-trained models using transfer learning and freezing the layers of the pre-trained model unlike model 2 and 3 which do not freeze the pre-trained model layers. The specific model used is a benchmark model not from the torch models but a high scoring CIFAR-10 classification model. The model variant is called densenet161. We use this model to initialize out model but freeze the layers, we experimented with adding our own layers but found a single linear layer at the end that does the class outputs gave the highest accuracy. Another change we made to this model is the optimizer used, rather than the Adam optimizer, based on readings regarding the best optimizer for the CIFAR-10 problem, we used the Adamax optimizer with a learning rate of 0.001. The loss was still calculated used the cross entropy loss. This model was trained and tested with the train and test data. The evaluation as primarily done through accuracy, precision, recall and f1 score on the test data as well as a confusion matrix output. Other details regarding this model include, the training was done

on 5 epochs to reduce training times given we freeze prior layers and given a single layer is all that needs to work through the data now, this count of epochs was sufficient. A GPU was used for this model as well.

e. Model 5:

Model 6 follows the same data setup as model 2. The only technique we hadn't explored yet was the adaptive learning rate. To experiment with this technique, we used the best model we had so far which is model 5. The way the learning rate was made adaptive was by using a scheduler from the optim library, with a step size of 5 and gamma of 0.1. By increasing the epoch count to 15 on the training epochs, what essentially happens is that every 5 epochs the learning rate is dropped down by a decimal place (0.001, 0.0001, and so on). With 15 epochs we saw the loss for each epoch at 4 levels of learning rates and post training, the model was evaluated using the accuracy, precision, recall, f1 score and a confusion matrix plot. The point to note is, aside from the scheduler, the model architecture is the exact same as model 5, as we wanted to know how much benefit a scheduler gives by isolating this technique rather than making further changes to the model which might abstract interpretability as to where the improvement or no improvement stems from. A GPU was used for this model as well.

5. Experiments.

Model 1 Outputs:

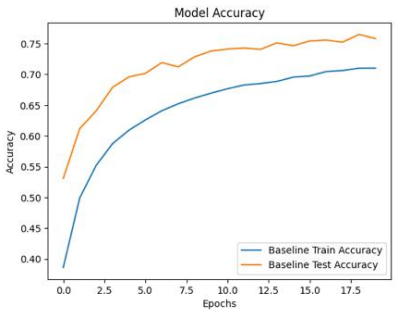

Figure 1. *Baseline Model Accuracy*

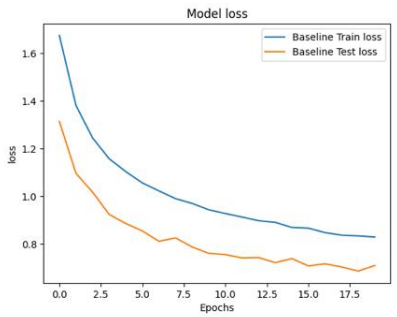

Figure 2. *Baseline Model Loss*

Model 2 Outputs:

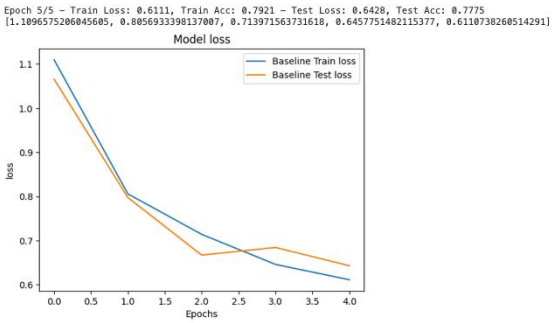

Figure 3. *Resenet18 Model*

Model 3 Outputs:

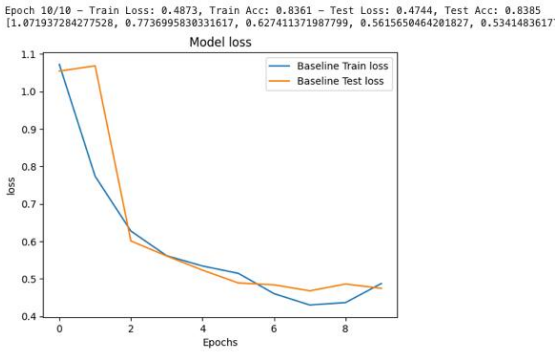

Figure 4. *Densenet121 Model*

Model 4 Outputs:

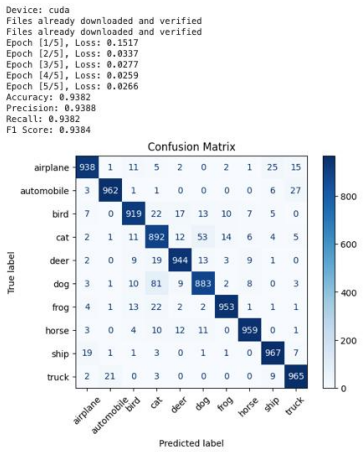

Figure 5. *Desenet161 - LR 0.001*

Model 5 Outputs:

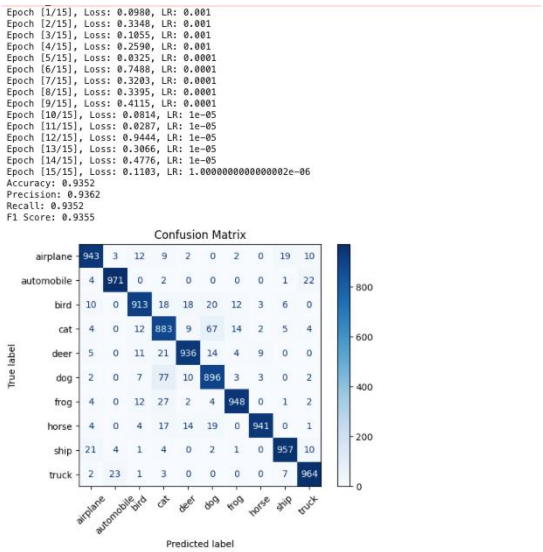

Figure 6. *Densenet161 - Adaptive LR*

6. Conclusions. In conclusion, of all the models that we trained, DenseNet seemed to perform the best, with DenseNet-161 performing well at 93% accuracy.

This can be attributed to the fact that the DenseNet-161 is able to explore features on a deeper level than the DenseNet-121 model, due to the dense block architecture being tailored for CIFAR-10. Additionally, the parameter count for DenseNet-161 is higher than DenseNet-121.

This makes it much more computationally intense than DenseNet-121, which would be reflected through longer training time. However, given that we are using transfer learning rather than doing it ourselves, especially considering that we are freezing the layers, computation was not a big problem. We also made use of a GPU to deal with any excessive computation time.

7. References.

1. **Huy Phan.** (2021). *huyvnphan/PyTorch-CIFAR10 (v3.0.1).* Zenodo. [https://doi.org/10.5281/zenodo.4431043]

2. **Cifar-10 Classifier.** (n.d.). [https://neelesh609.github.io/cifar10/]

3. **Nguyen, T.-T., & Yoon, S.** (2019). *A Novel Approach to Short-Term Stock Price Movement Prediction using Transfer Learning.* Applied Sciences, 9(22), 4745. [https://doi.org/10.3390/app9224745]