Chenchimin Lee
Jared Vue
Jonathan Xiong
CSCI 167 Intro to Deep Learning

# Deep Learning Model for Image Classification

**Github Link for ipynb:**
https://github.com/CxxxLee/167-Project/blob/main/Project167.ipynb

## *Objective:*

The goal of this project was to develop a deep learning model that can classify images into predefined categories. The task involved training a Convolutional Neural Network (CNN) to recognize patterns in the images, followed by model evaluation and prediction on new, unseen data. The project provided an opportunity to explore various deep learning concepts, including dataset preparation, model creation, training, hyperparameter tuning, and model evaluation.

## *Methodology:*

### Dataset Exploration and Preprocessing:

The dataset used in this project was the Intel Image Classification Dataset from Kaggle(https://www.kaggle.com/datasets/puneet6060/intel-image-classification/data), which contains labeled images of the following classes: 'glacier', 'forest', 'mountain', 'sea', 'street', and 'buildings'.

The dataset was divided into training and testing sets, with image augmentation techniques like rotation, shifting, zooming, and horizontal flipping applied to the training set to improve generalization and reduce overfitting. The dataset had a third set called prediction which had images we ran on the model. Images were resized to a uniform size (150x150 pixels), and pixel values were normalized to the range [0, 1].

### Model Architecture:

The chosen model was a Convolutional Neural Network (CNN). The model consisted of three Conv2D layers, each followed by MaxPooling2D layers to downsample the feature maps. The number of filters in the convolution layers range from 32 to 64 then to 128. The model includes a Flatten layer to convert the 2D feature maps into a 1D vector. Dense layers with ReLU activation and a Dropout layer helped prevent overfitting. The

final layer used a softmax activation function to output the predicted class probabilities for multi-class classification (6 classes in total).

**Model Training:**

The model was trained using the Adam optimizer and categorical cross entropy loss function, as it was a multi-class classification problem. The model was trained for 1 epoch initially, (to test if the model worked) then we increased epochs in future iterations to allow the model to learn effectively, increasing the accuracy of our model and reducing the loss. We found that a higher number of epochs lead to a higher accuracy for our model but due to hardware constraints we weren't able to test any epoch numbers higher than 20. The training time took too long with higher epochs. We also messed around with batch sizes as well. We saw that smaller batch sizes lead to very long training time as well. So we ended up with a 32 batch size for our training generator and 64 for our test generator. Validation data was used to monitor the model's performance during training, ensuring that the model did not overfit to the training data.

**Performance Evaluation:**

After training, the model was evaluated on the test set using accuracy as the primary metric. Additionally, a confusion matrix was plotted to visually assess the model's performance, identifying areas where the model misclassified images. The model achieved a good accuracy of 80 - 90 % on the test set on different training attempts around 10 - 15 epochs with the batch sizes from before, but further improvements would need more hyperparameter tuning or model adjustments.

**Prediction:**

A function was implemented to predict the class of a single image. This function preprocessed the image, passed it through the trained model, and returned the predicted class label. The prediction function was tested on various test images, and it identified the correct class on a lot of the images.

## *Key Findings:*

**Model Performance:**

The CNN model had reasonable accuracy on the test set. However, it can improve, especially in cases where the model got confused by images that were more ambiguous. The confusion matrix revealed classes that were often misclassified like seas and glaciers or streets and buildings. These classes are similar to each other and often have components of each other in the images. This leads to our model being confused on what to label it as which causes our model to be more inaccurate. Our

results also vary from each iteration due to different hyper parameters and different results from training. An example of this is in one run our model was very efficient in predicting mountains but not very efficient in forests. However, in a different run, it was the opposite. Outside of these types of situations, our model is reasonably accurate though it can be improved.

**Data Augmentation Impact:**

The use of data augmentation techniques significantly improved model generalization, reducing overfitting and helping the model learn more robust features from the training images. Image augmentation enabled the model to recognize objects from different angles and positions, making it more adaptable to real-world scenarios.

**Hyperparameter Tuning:**

While the model performed decently with our default settings(0.5 dropout, 10 epochs, batch size of 32 for training data, batch size 64 for test data), experimenting with hyperparameters (the number of epochs and batch size) could potentially increase or decrease performance. In this case we consider the time it takes to run the model as a part of the performance alongside the time it takes to run the model tasks and the hardware usage.  We noticed that large amounts of epochs (20+) resulted in extremely long training times. (2 - 3+ hours of training). We also noticed that smaller batch sizes for our data resulted in extremely longer training times.

**Prediction Efficiency:**

The prediction time for a single image was fast. Showcased more by the live demo since the live demo tests the model with images it wasn't trained on. `Prediction completed in 0.1281 seconds` was one of the time results we had with the live demo section. Our model could be more accurate which reduces the efficiency of our model but it is accurate enough.

## *Conclusions:*

**Model Strengths:**

The CNN model was effective for image classification, especially with the appropriate preprocessing, data augmentation, and architecture choices. The model's ability to make predictions on new images, images grabbed from google (with size 150x150), successfully demonstrated the model's potential for real-world applications. The model can be very efficient and is accurate enough to identify and classify an image into a category we expect it to without it being in the dataset.

**Limitations:**

There is a lot of room for improvement. The model's performance could be enhanced by fine-tuning hyperparameters like increasing epochs or optimizing batch size logic. The confusion matrix revealed some misclassifications between certain classes, suggesting that additional data or more sophisticated models might be needed to distinguish these classes more effectively. It also suggests that our model needs more fine-tuning in order to distinguish the differences in these edge cases where the images are more ambiguous. The loss and accuracy graphs also revealed an interesting point for our model. As seen in the graph for accuracy we saw that the validation was higher than the training and in the loss graph we saw that the validation loss was lower than the training loss. We concluded that the cause of this was the testing/validation data only having around 3,000 images compared to the training data having around 14,000 images. This most likely caused the training data to be more noisy and causing it to have a lower accuracy and higher loss.

**Future Work:**

Future work could involve experimenting with live images and footage. So that the model can classify a location under its class constraints in direct real time.  Further exploration into hyperparameter optimization could help identify the best configuration for the model. This model is a good basis to build upon. It can always become more complex and more optimized.

**Applications:**

This model has several potential applications in fields like automated image labeling, environmental monitoring (e.g., identifying types of land coverage), and even in areas like autonomous vehicles, where real-time image classification is necessary. The model can be deployed on devices for real-time image recognition in various industries.

**Group Contributions:**

We all did the same amount of work. It was a group effort.

**Works Cited**


Kaggle - https://www.kaggle.com/datasets/puneet6060/intel-image-classification/data
Dataset used for the model

Deep Learning Book - https://udlbook.github.io/udlbook/
For understanding and reference on making a model and deep learning

ChatGPT - https://chatgpt.com/
For help on understanding and finding resources

Tensorflow Documentation - https://www.tensorflow.org/api_docs/python/tf
For learning and better understanding tensorflow and Keras

Seaborn Documentation - https://seaborn.pydata.org/api.html
For confusion matrix graph

Scikit-Learn Documentation - https://scikit-learn.org/stable/api/index.html
Confusion Matrix

Matplotlib Documentation - https://matplotlib.org/stable/api/index
For graphs