Deep Learning

Multi-Classification Image Project

Team 10
Joshua Pauly, Josue Perez Gomez

School of Graduate Professional Studies
Data Analytics
DAAN 570 – Deep Learning
Fall, 2023

Document Control

Work carried out by:

Name	Email Address	Exhaustive list of Tasks
Joshua Pauly	Jfp5940@psu.edu	Transfer learning models
Josue Perez Gomez	Jkp5482@psu.edu	Data Analysis and vanilla CNN approach.

Revision Sheet

Date	Revision Description
11/29/2023	Revisiting Content and adding to sections.
11/29/2023	Adding graphs, images and more content.
12/1/2023	Added more graphs/images and more content.
12/2/2023	Added more content and organized a bit more.
12/7/2023	Added more graphs and content
12/8/2023	Finish paper

TABLE OF CONTENTS

1 INTRODUCTION 3

2 PROBLEM STATEMENT 3

2.1 CHALLENGES 4
2.2 RELATED WORKS 4
2.3 IMPORTANCE 6

3 DATA COLLECTION 6

4 DATA PREPROCESSING 8

5 METHODOLOGY 11

6 RESULTS 14

7 DISCUSSION OF RESULTS 17

8 REFERENCES 19

1 INTRODUCTION

[Computer vision has been a field of study for a long time since around the 1960s. Followed by the 1970s many new algorithms were developed that could detect edges, corners, and the use of filters, optic flow algorithms to detect different features. Thanks to these methods it has been possible to identify objects within an image, such as a person, animal, or a car. However, trying to accurately identify an environment or natural scenery is a bit more difficult.

This is why the objective of our project is aimed at achieving a high accuracy and precision in an image multi-class classification problem. A classification problem where we aim at identifying the overall scenery of an image, instead of a specific object within the image. Additionally, most professional cameras do not have a method to identify the type of location the image is taken of. Which can be helpful in organizing and sorting images if the user would like.

For this project we aimed at exploring different techniques and models that can help in solving this problem. Along with it, explanations to each solution and how we determined the success of it. This report also concludes with a summary and conclusion of our work on this image multi-class classification problem.
]

2 PROBLEM STATEMENT

[The problem that our project is trying to tackle is to come up with a solution to an image multi-class classification problem as explained in the introduction. That is, the data that we used were images.

For the purpose of this course, we aimed at building a Deep Learning based model that was capable of accurately and precisely identifying the type of natural scenery present in each image of the dataset. This is with the goal of having a model that could take in new never seen images as inputs and the model successfully returning an output that would be the label the model best associates the new image with. This only after learning how to distinguish images into the six different categories.

Some issues that come with this image multi-class classification problem revolve around the data and the architecture that our deep learning-based model would be. As well as the tunning of the hyperparameters and any type of preprocessing done before training a model. For instance, right away during the project proposal we noticed that the data that we would be working on had a storage size of around 368.86 MB, which was a bit of an issue when not too much RAM storage was available.
A potential problem we mentioned in our proposal was the possibility that there would be too much variation in the images, even between images that belonged to the same class type. We thought that it might've had some effect on not getting a desired accuracy score. This was something the team had a concern about since it would mean we would have to spend more time than expected on coming up with solutions on this image variation. Luckily, as we will show in

our results and interpretations, we noticed that the variance between images did not have as much of an issue as we anticipated.

Additionally, a problem we identified earlier on was the ability or gap of being able to work with all of the images available for training. This actually became a real issue when we tried to work on the Deep Note site for one of our notebooks. As a result, we had to save checkpoints during our data manipulation, filtering and dimensional reduction so we could work with as much data as we could. For this notebook we were able to work with around 8,100 images out of the 13,000 available for training.
]

3 CHALLENGES

[For this project there were some challenges that we had to deal with. One such challenge was deciding how we were going to approach the task. That is which methods, algorithms and type of Deep Learning architecture we were going to use. For this we made use of the notes from Canvas and the example notebooks published on Deep Note.

We had a few challenges with the data itself. Mostly making sure we read them and had them stored in data types that were useful for us to work as. Though, this was a relatively easy challenge to tackle thanks to the large number of libraries that are available in Python. Thus, our data challenge was an easy one to solve.

Initially we expected to have challenges with the predictive analytics, given that there was a lot of variation in the types of images. For instance, while looking at the images we noticed that an image of a glacier might get labeled as being an image of the sea, since a glacier is usually near a body of water. This we thought could throw off the model into giving the wrong prediction for an image. During evaluations of the model, we did notice that the models were having a hard time properly labeling the glacier images as glaciers. However, they were not being wrongly labeled as sea images, but instead as mountain images. This is why we tried to do data augmentation.

As far as the business challenge on the other hand. The challenge was in coming up with a model that was up to a good standard and accuracy. The reason was because if we were to implement a model or multiple models into a software that could sort new images from a client into their proper classes, we must make sure that the model could accurately sort them as frequently as possible. Otherwise, the software or cameras that make use of the models would have a bad reputation for having an unreliable image sorting feature.
]

3.1 RELATED WORKS

[The data set that we used in our project was provided by Intel as part of an image recognition competition they had back in 2018.

Some related projects that try to do image recognition of the entire image is the handwritten number recognition project(s) from the infamous MNIST dataset. Many people have created models that have been able to recognize the number in an image from the MNIST dataset. However, given that the images in that data set had dimensions of 28x28 pixels, it meant that a simple Multilayer Perceptron model could solve this problem. The same approach would not work with the data that we used. Given the images had a 150x150 pixel dimension. Therefore, we had to pick and use a different deep learning architecture to solve our image classification problem.

For instance, in one paper ([1] Yu, Tang, & Li - 2013) the researchers attempted to solve an image classification problem on the MNIST dataset, however instead of using CNNs, data augmentation, or normalization. The researchers instead made use of a temporal encoding preprocessing before using a MLP architecture for their model.

Another team of researchers tried to solve a face recognition problem. Here the researchers tried to solve a similar problem to ours by trying to correctly predict the class belonging to an image. Whereas in our project we worked with natural scenery and general environment, in ([2] Wen, & Shi - 2007) this related project they worked with images of faces. Thus, working at classifying the class of a whole image. However, the biggest difference is that we made use of Deep Learning models while in ([2] Wen, & Shi - 2007) this other team they worked with a modified version of PCA, by looking at the components that could explain most of the data and then used the most important ones to help them in recognizing a face.

Another team ([3] Bhowmik, Bhattacharjee, Nasipuri, Basu & Kundu - 2009), similarly to ([2] Wen, & Shi - 2007) the first team tried to perform face recognition research. Also similar to ours, they aimed at identifying an image and correctly classifying it. However, this was different again because they made use of PCA as well, except they also made use of neural networks. Also, they did not make use of a CNN architecture for the model. What this team ([3] Bhowmik, Bhattacharjee, Nasipuri, Basu & Kundu - 2009) separated from team ([2] Wen, & Shi - 2007) was that they made use of fusing images that were taken with thermal cameras instead of just using images in the visible spectrum.

A fourth team ([4] Georgiadis, Cavouras, Daskalakis, Sifaki, Malamas, Nikiforidis & Solomou - 2007) aimed at building a personalized digital assistant to help with discriminating three major types of brain cancer. However, this team made use of a probabilistic neural network which was a four-layer feed-forward neural network classifier. Their model made se of a probabilistic density function that was in charge of discriminating each image to their respective class. This research differs from ours since they made use of a feed-forward neural network with a PDF to discriminate images to detect the type of brain cancer. While ours made use of a CNN to distinguish the type of scenery in images.

Lastly, another team ([5] Hussain, Bird, & Faria - 2019) wanted to explore the use of transfer learning model Inception-V3 previously trained on ImageNet. The experiments took place on two datasets CIFAR-10 and Caltech Faces. Compared to previous studies of training models trained from scratch scored an accuracy rate of 38% whereas the transfer learning accuracy rate was 70%. It was clear that in the study even given limited computing powers transfer learning from inception-v3 high accuracy scores when compared to the scratch models results. Given this

information we would like to incorporate both methods a scratch model and multiple transfer learning models to compare results.
]

3.2 IMPORTANCE AND IMPACTS

[We think that the importance of this project is to show a complete process in creating deep learning-based models for identifying natural scenery. This is something we think is important to cover, given that most image recognition research and work has been focused on locating objects within an image or in face recognition. Whereas what our research focused on was different natural scenery and human structures. Which can potentially have a less intrusive application for society, than a facial recognition model or tool.

Currently our project and the results have the potential to be incorporated into a software that can take many images and sort them based on their scenery. For instance, when people go out on trips and take pictures (especially with professional cameras) there exists the possibility that the pictures are taken at different times for the same location. Which could result in a handful of pictures of a forest being in between pictures of buildings only to have more pictures of a different or similar forest at the end of the pictures of a street. At the end of a trip if a person is to save his or her pictures in their computer, if they want to save all of their forest images into a folder, they would have to manually move them. Which can be very time-consuming.

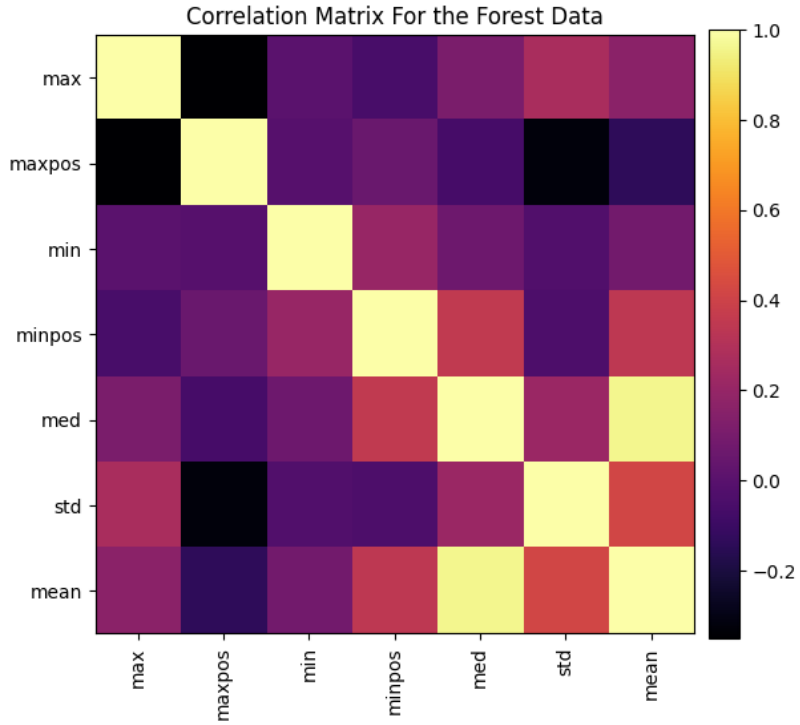
Here is where our project impact can come into play. By passing the location of the folder that has all the images to the software or allowing to have access to it, the tool can sort the pictures by the categories it can recognize. Or have the user to enter the desired classes they want their images to be sorted into. This type of tool can save the owner of their pictures time sorting them into folders and keeping them more organized. Not just that but since our models only require the model architecture and the weights, besides the Python interpreter, there is no need for the use of any internet connection to sort the images by their corresponding class.
]

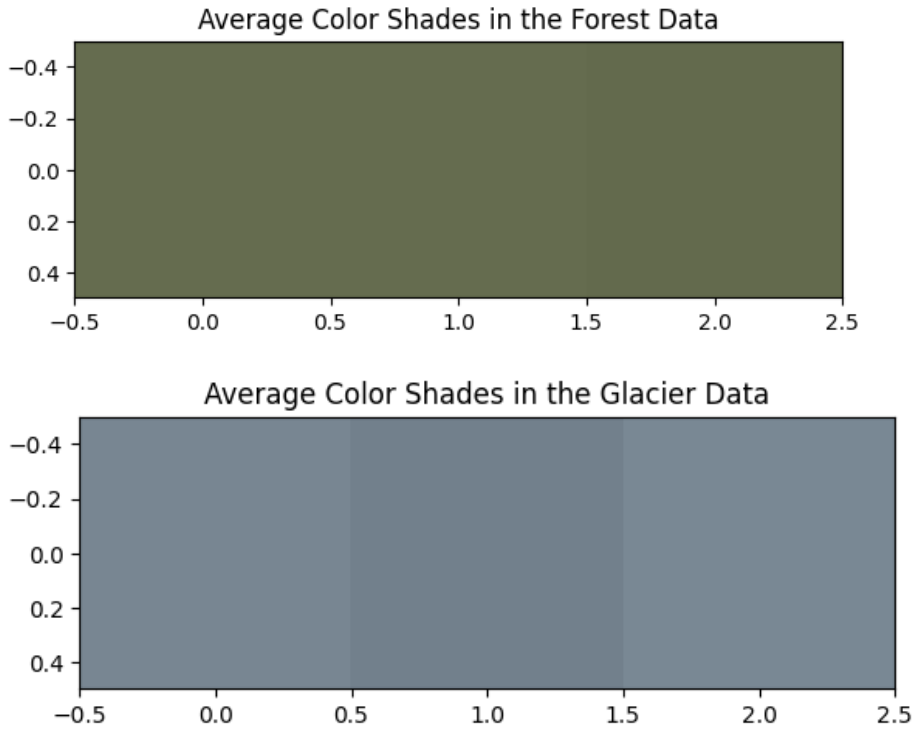
4 DATA COLLECTION

[As mentioned, the data that we used for this image multi-class classification problem came from an Intel image recognition competition. Thankfully a Kaggle data set repository was created and contained all the data used during that competition. This made it easy for us to simply download the data that was already sorted into a test and training folders, as well as the images that had been sorted and grouped into folders for each of the classes they belonged to, such as folders for mountain, building, sea, street, glacier, and forest images. All of this made it really easy for us to get started. Additionally, given that the data was published in an open-source repository in Kaggle it was very much permitted to be used, according to Kaggle's own rules. The data itself came from a 2018 Intel Scene Classification Challenge, which was open for the public to use as well.

Once we got our data, we first did a data exploratory analysis. For this project we worked with images, so our data was of a categorical type since it is not meant to represent any numerical representation, like a continuous or discrete variable.

Given that the data we were working with were just images, some of the data exploratory analysis we could do was on the RBG representation on the images. Once we had the images in this format, we were able to do a max, min, mean, standard deviation, 1st, 2nd and 3rd quartile analysis on the colors in the images. Additionally, we wanted to see what shade of color was most present in the images for each class. We did this by using the KMeans algorithm to get the clusters from the three channels (red, green, and blue). Which then we use that to multiply it by the standard deviation divided by 255. Finally, we computed the mean of the values in each channel for each image to get an idea of the average color shade of colors in the images per class.

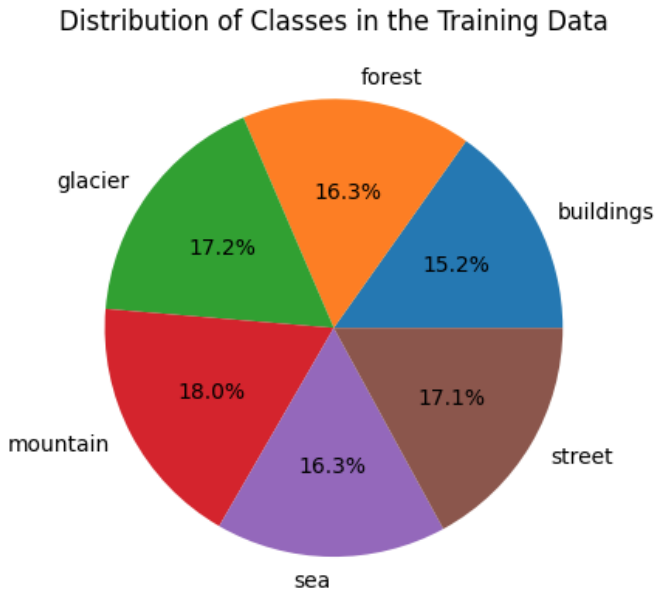




We noticed that there was a no major correlation between the statistical values we computed for each class, except for the median and mean values for each class. Which is something we expected from. And we did notice as expected that each class of images would have their own distinct shade of color for their red, green and blue channels.
]

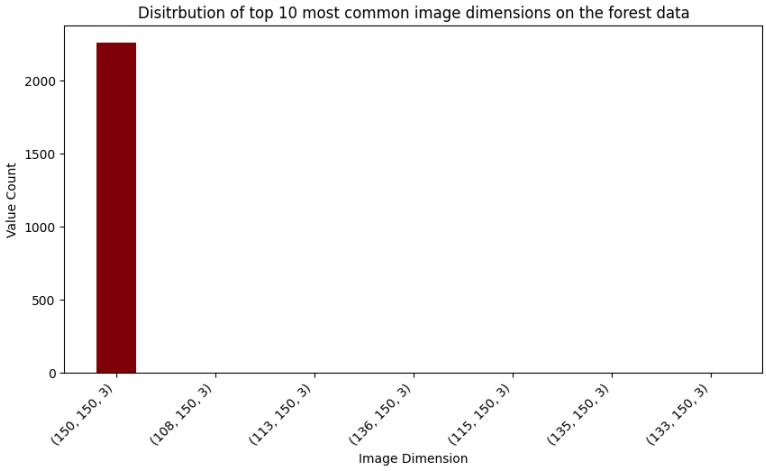
5 DATA PREPROCESSING

[Given that our data consisted of images we were only able to generate a few statistical descriptions on the data. For example, a descriptive statistic that we were able to generate was a frequency distribution of the classes in the training and testing data. As well as the already shown correlation matrix on some statistical computations for each class.

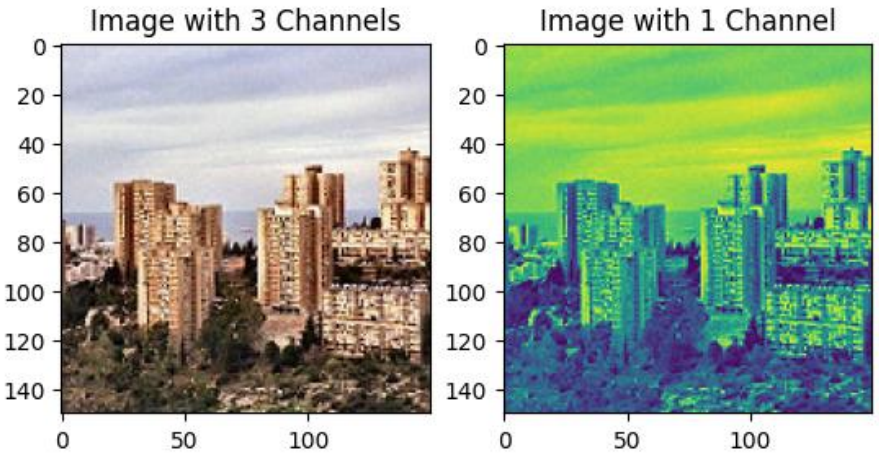


One way we got the images loaded into our notebook was by getting the directory of the images and then literately reading them one by one. We used the **mpimg.imread** function in the matplotlib library. This process gave us an array of data type ndarray, all the images had a 3-dimnesion shape, one for the width, length and number of channels. We then converted it into a **NumPy** array with the same dimension shape and append all of them into a list for that given class.

Before we did any sort of transformation and normalization, we investigated any possible missing values of inconsistency. For that we wanted to look first at the distribution of the different dimensions in the images, and we noticed that the vast majority of the images per class had a dimension of 150x150x3. With less than 1 percent of the data having a different dimension.



So, we decided to exclude the images that did not have a 150x150x3 dimension. Thereby avoiding scaling the data or normalizing it. Normalizing the data is a required step, since regardless of the architecture we decided to use, they all would have to have an input shape defined. So, it was needed to ensure all the images could satisfy the initial input shape. What we did however was to reduce the dimension of the images from 3 channels to 1 channel, that is from an RGB to a black and white image. We did this by using the **color** function in the skimage library, again one by one. For example, this comparison of an image with 3 channels compared to 1 channel.

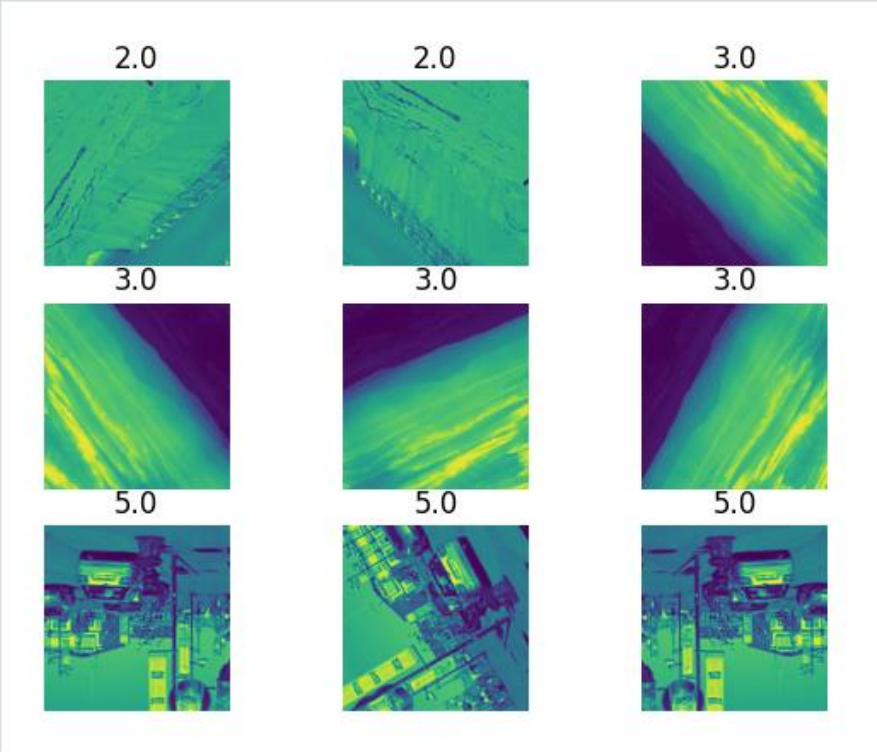


After looking at the class distribution in the data, it was clear that there was no significant imbalance between the classes in the data. And after looking at the shape distribution there was no real inconsistency with the data as well.

Given that this was an image multi-class classification problem, the metric that we used to interpret our results was via calculating the accuracy of the labeling that an image would get from our model. Additionally, to complement the accuracy metric we made use of a confusion matrix to get a more insightful result of our data.

Additionally, we gather graphs of the accuracy and loss values during model training to visualize if there was a timestep where learning was not happening, or accuracy stagnated. The way we interpreted our data was based on how well learning was done and the scores when calculating the accuracy scores for our test data. Supported with a confusion matrix to identify which classes the model had a harder time identifying correctly.

We also performed some data augmentation on our data. Data augmentation can help in generating new data and for our specific image multi-class classification problem data augmentation can take images from the training set and produce altered versions of them. These transformations consist of rescaling, resizing, rotating, zooming, flipping, and changing the brightness.



Example of data augmentation using rotation on images of glaciers, mountains, and streets

Rescaling an image size of 224 x 224 allows CNNs outputs feature maps of size 7x7. This is these CNN have learned to find certain patterns of certain sizes, so in transfer learning it helps to resize to what a model was previously trained on. ([6] Koehrsen - 2018) In our case our transfer learning models were trained on 224 x 224 images. So resizing the images gives us a unique advantage when training transfer learning models as they are created to pick up on patterns of that size.
]

6 METHODOLOGY

[
The methodology that we used in our project were data exploration to verify that the data we worked with did not have any skewness in class representation, we also used dimensional exploration and reduction to gather images that could fit our model inputs and RAM capacity. As mentioned above some of these methods were transforming our images that were in an RGB array representation into a single channel representation, that is in black and white.

For the choice of models, we tested with, they all made use of the Convolutional Neural Network architecture. This architecture makes use of convolutional layers, max pooling and/or mean pooling layers, dense layers and in some cases a dropout layer. The reason we decided to use this type of architecture was because Convolutional Neural Networks (CNNs) make use of convolutional layers. These convolutional layers work by ([7] Zhang, & Yu - 2020) passing using what are known as a convolutional kernel or filter which performs a dot product operation on a section of the image. Which produces an output that later can be used to compute the max-pooling

or average-pooling, this is to take advantage of nearby pixels that are similar, which can also reduce the dimension. This reduction can make the following convolutional layers more efficient.

As mentioned, our convolutional layers need to have a cost function, activation function and weights and biases to it. For our cost function we used the sparse categorical crossentropy loss function, which is a loss function that computes the cross entropy between the labels and the predictions. This loss function is mostly used when there are two or more label classes, which makes it perfect cost function for our image multi-class classification problem. This cost function is a variation of the categorical crossentropy function, except that sparse categorical crossentropy is best when our categories are mutually exclusive and when our targets are as integers instead of in a one-hot encoding. In our case the label data was in integers, so it worked perfectly.

We used different activation functions for different layers. For example, for our convolutional layers we used the ReLU activation function. And for our output dense layer we used a SoftMax activation function. The Relu activation function works by adding non-linearility to our deep neural network model, additionally ReLU is easy to implement and compute in compared to a sigmoid or tanh function. As for our SoftMax activation function, it is used to provide some normalization, and generally is used on the output layer since it can normalize a vector into a probability vector which makes it great for classification problems. ([8] Chollet - 2018)

As far as weights, CNN makes use of parameter sharing, which allows for parameters to be shared across different steps in the model. This in turn reduces the number of parameters needed to keep track of. And this is possible since in convolutional layers, it makes use of filters which essentially act as weights in the network.

After exploring the use and work of a CNN there are many strategies to validate the models we built. This is where we created two different strategies, one using vanilla CNN and the other using transfer learning followed with some fine tuning on our data.

For our vanilla CNN the architecture for our best models consisted of the 4 convolutional layers, 4 max-pooling layers, 1 dropout layer and 3 dense layers. The dropout layer works as a way to include regularization that prevents co-adaptation of neurons ([7] Zhang, & Yu - 2020) by having some neurons randomly being deactivated.

For transfer learning-based model we made use of 4 models to compare the results. These models were VGG-16, VGG-19, Efficient Net, and Resnet 50. We chose these models because they are uniquely created in their architecture so we wanted to test how different structures trained from ImageNet would do against our data set, and all these models were able to be imported from Keras.

VGG stands for Visual Geometry Group. The original paper had two models of 16 and 19 layers depth, thus VGG-16 and VGG-19. ([9] Shanmugamani - n.d.) The VGG models comprise of convolution layers max pooling layers three fully connected layers and a SoftMax layer. Receiving a 224 x 224 image the convolutional filters of VGG use the smallest possible receptive field of 3×3. VGG also uses a 1×1 convolution filter as the input's linear transformation. All hidden layers in the VGG network use Relu activation. Within the structure pooling layers are used to reduce dimensionality by adding channels. After the first max pooling layer an image will reduce to a 224 x 224 by 3 channel to a 112 x 112 x 6 channels doubling the channels, but significantly reducing

the image size. Because of the additional three convolutional layers in VGG 19 we expected an increase in run time compared to VGG-16.

EfficientNet is a convolutional neural network that is based on the notion of "compound scaling." Compound scaling is intended to scale three critical aspects of a neural network: width, depth, and resolution. ([10] Ai, - n.d.) The number of channels in each layer of the neural network is referred to as width scaling. By enlarging the width, the model may capture more complicated patterns and characteristics, resulting in higher accuracy. Reducing the width, on the other hand, results in a more lightweight AI model that is ideal for low-resource applications. The total number of layers in the network is referred to as depth scaling. Deeper models can capture more complex data representations, but they also need more computing resources. Shallower models, on the other hand, are more computationally efficient but may forfeit accuracy. Resolution scaling entails changing the size of the input image. Higher-resolution images give more detailed information, which may result in improved performance. They do, however, need larger memory and processing power. Lower-resolution images, on the other hand, use fewer resources but may lose fine-grained information. Using this model even with compound scaling we expected it to run the slowest when training, but also expecting it to produce the best results, because it was by far the deepest model.

ResNet 50 took an interesting approach to it's architecture. The creator's original paper mentioned the researchers wanted a deep network, but when doing so the loss function was too high as the original input was being lost. To remedy this effect the model incorporates residual connections that allow for the identity function to be passed as an output without a penalty to the model. Because of the residual connection multiple layers can be viewed as residual blocks as they can pass their input signal along. This is one of the more popular models used today because of the reduced loss function. We expected this model to perform well and have quicker run times than Efficient Net but slower than VGG models as it has a depth of 50.

Of the transfer learning models we found an accuracy score roughly 90.5% across all four models, after training with 15 epochs and a batch size of 128. However, the run time trainings were not similar at all. EfficentNet with the most trainable parameters was the longest run time around 8 hours, then the VGG models were similar run times of around 4.5 hours and finally the Resnet 50 model had a run time of only 4 hours outperforming our expectations. With similar accuracies we explored Resnet 50 further with more experimentation finding increasing the batch size to 512 increased the accuracy to 91.1%.

Based on these two approaches we came to the conclusion that overall, our models that were built using transfer-learning had a higher accuracy score than a vanilla CNN that did not had transfer learning. However, the drawback was the amount of time it took to train the models, which was many more hours than the vanilla CNN took. For comparison our best vanilla CNN model attained a test score of 78.85%, whereas our model that had transfer learning had a test accuracy score of 91.1%

However, comparing both of our approaches to traditional machine learning it is clear that even a vanilla CNN can attain better results than a feed-forward neural network that does not make use of convolutional layers and pooling layers. Even more so having transfer learning into our models

attain scores that a feed-forward neural network could not reach. As mentioned early on, traditional approaches would make use of PCA and then a feed-forward neural network. However, PCA can be very computationally expensive when there are many variables and data. In contrast convolutional layers and pooling layers do not have to keep track of many parameters since it makes use of filters that can be shared among layers.

Early on the team had assumptions about how good the models would perform and we were worried that it would need a lot of tunning to create good models. Initially, this was not the case since a single convolutional layer CNN managed to get an accuracy of 60%, however after we modified our model, it became harder to improve our accuracy scores.

All of this was possible thanks to Python and its large libraries that Python and other communities have put out for people to use via open-source libraries. As well as pretrained models like VGG16 which we used for our transfer learning approach that we fine-tuned to our data. Additionally, we use Jupyter notebooks to organize our work into sections that we could better break down our work.
]

7 RESULTS AND INTERPRETATION

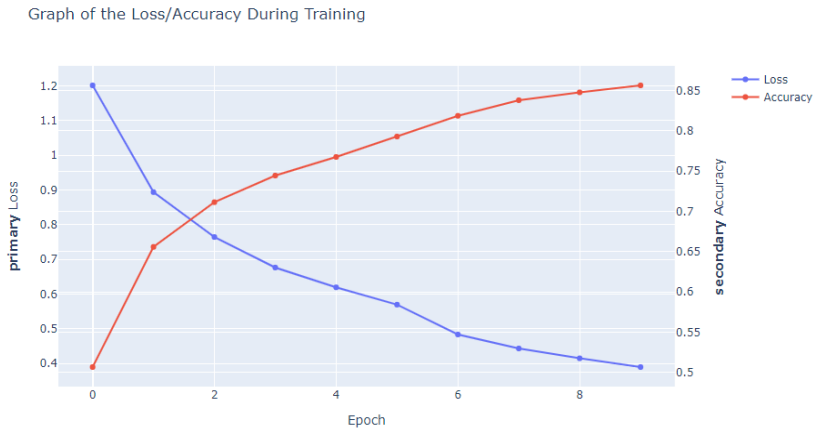
[For this problem we built and trained many models, saving them and their weights for future use and evaluation. The outcome of the models was evaluated by how well the accuracy got, and more importantly the accuracy score on the test data, which was what really mattered.

We found that the accuracy of the transfer learning models were the same after the same training conditions addressed in the methodology section at 90.5%, however the accuracy score did increase for resent 50 to 91.1% when changing batch size to 512 and increasing epochs to 25.

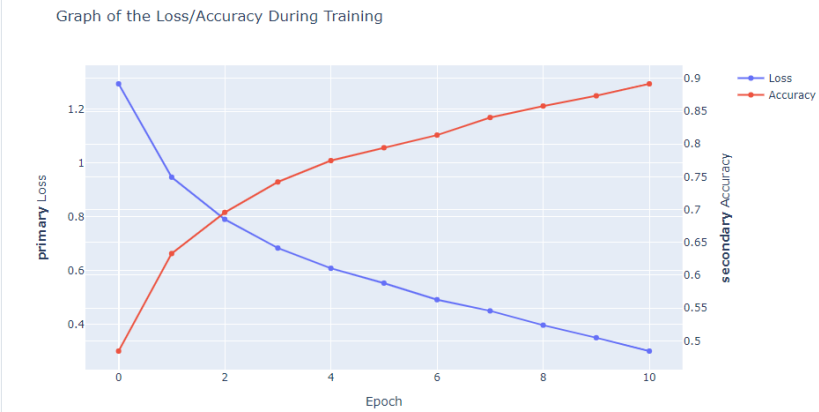
The results for the transfer learning models shows a lower training score than validation or test data. This is because of data augmentation made these images harder to predict than the original images.

On the other hand for the vanilla CNN models we built the mode that had the best test accuracy score was one with 3 convolutional layers with a kernel size of 48 and 1 convolutional layer with a kernel size of 64. Each followed by a max pooling layer that had a size of (2,2). After that we had a dropout layer that added regularization. Followed by 1dense layer with 125 neurons, another dense layer with 64 neurons and finally another dense layer with 6 neurons, each representing the different categories of images. This model had an accuracy score on the training data of 91% and an accuracy score on the test data of 78.44%.

We also tried to make use of the augmented data on the best vanilla CNN model but we noted that the accuracy score of the test data was closer to 60%.

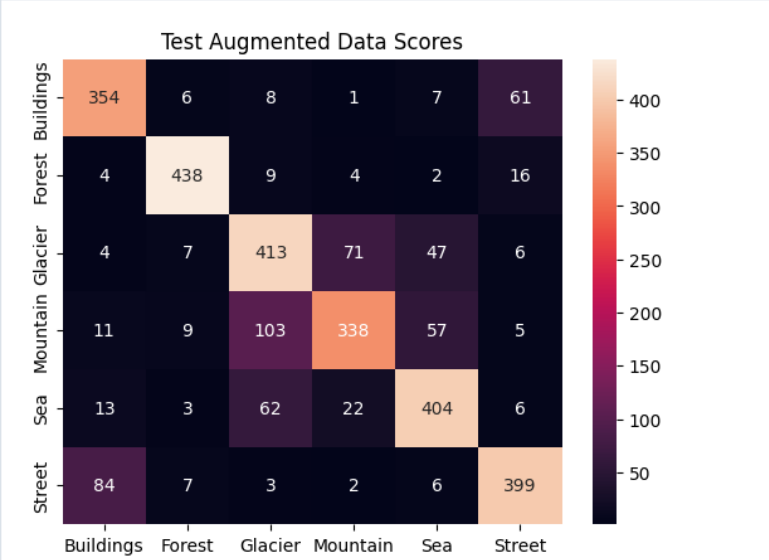


Graph of the loss and accuracy values for a vanilla CNN model that had a test accuracy score of 75.46%



Graph of the loss and accuracy values for a the best vanilla CNN model that had a test accuracy score of 78.66%

Here is a confusion matrix on the accuracy score on the test data for the best model.

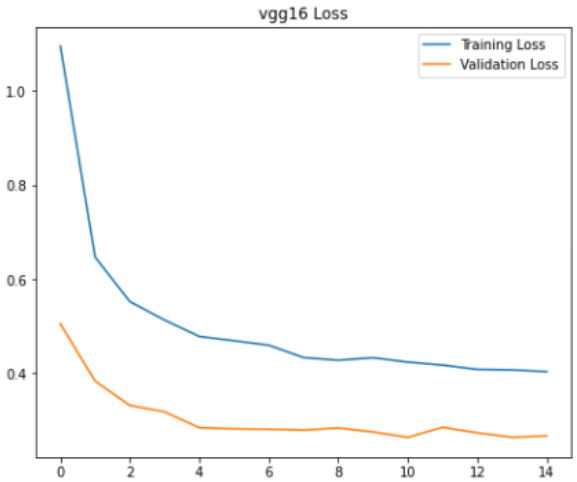


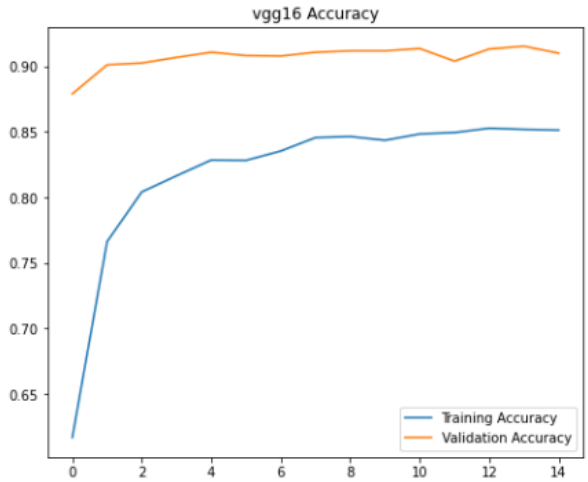

```
tf.Tensor(
[[354   6   8   1   7  61]
 [  4 438   9   4   2  16]
 [  4   7 413  71  47   6]
 [ 11   9 103 338  57   5]
 [ 13   3  62  22 404   6]
 [ 84   7   3   2   6 399]], shape=(6, 6), dtype=int32)
0.7840909090909091
```

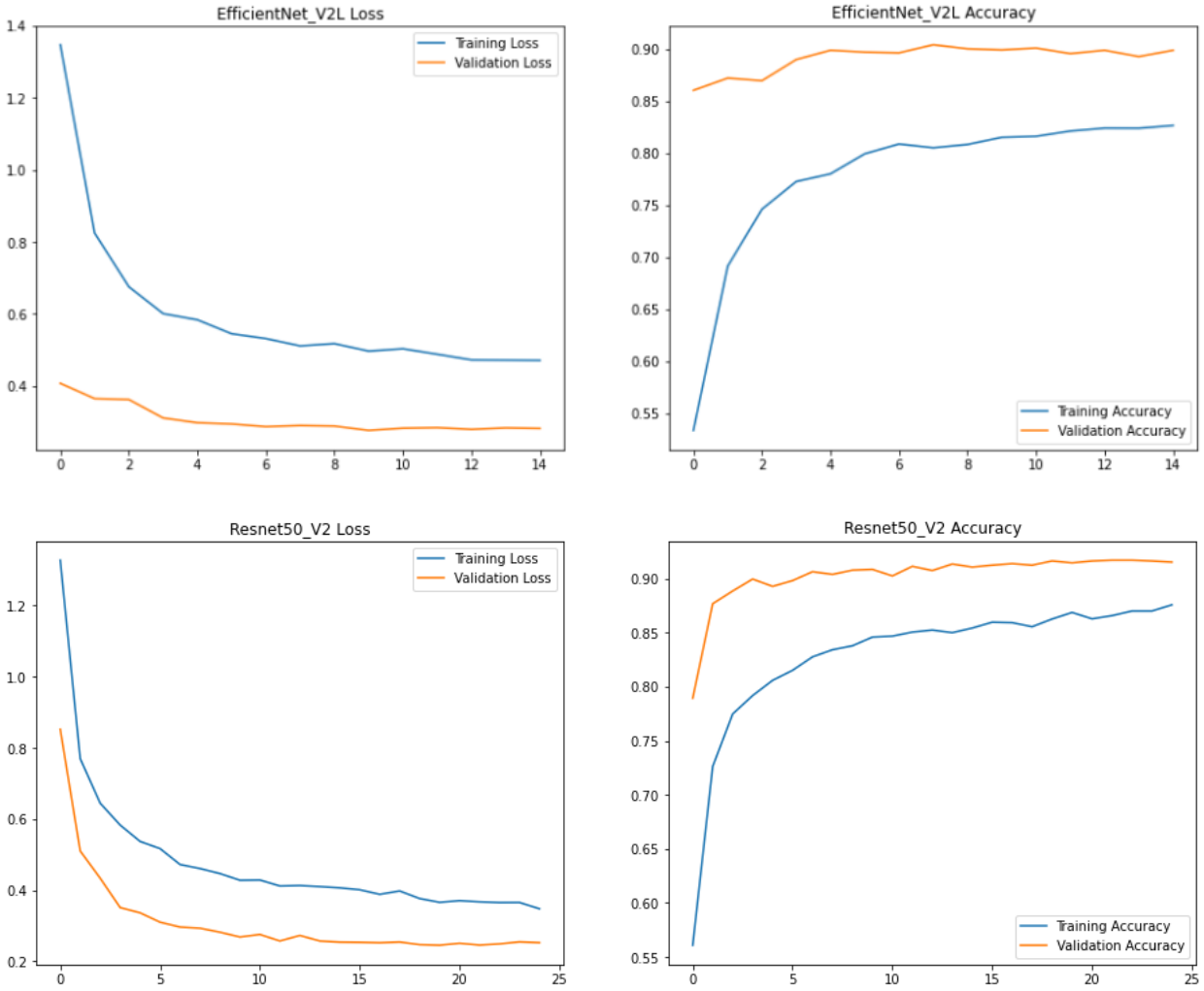
We also computed the precision and recall values on the confusion matrix on the accuracy score on the test data for the best model. We noted that the values where somewhat close to 1, with scores for the precision and recall on the glacier and mountain images having the lowest.

```
Precision: [0.7531914893617021, 0.9319148936170213, 0.6906354515050167, 0.771689497716895, 0.7724665391969407, 0.8093306288032455]
Recall: [0.8100686498855835, 0.9260042283298098, 0.7536496350364964, 0.6462715105162524, 0.792156862745098, 0.7964071856287425]
```

The reason why we think the recall, precision and accuracy scores on the glacier and mountain images are because images of glaciers tend to have a shape similar to that of a mountain which can be why the model confused most of the images between the two.







Transfer learning models accuracy and loss function over Epochs

	loss	accuracy
Resnet50_128	0.263767	0.901333
Resnet50_32	0.262779	0.903333
Resnet50_512	0.246893	0.910667

Accuracy rate on the 3,000 images in the test data set from different training batch sizes.
]

8 DISCUSSION OF RESULTS

[Having presented the result from the project, we believe that the results were aligned to what we expected. We knew that working with such a large and diverse data set would cause a challenge in generating a model that had a good score, especially with the approach with the vanilla CNN.

Some of the practical implications of the results are that by using our model that made use of the transfer learning showed that we are able to produce good accuracy scores. Thereby a beta version of a software that makes use of this model can be launched for people to try and get feedback, which can help us fine tune our model even more.

Some of the limitations of our work are the same as most image multi-class classification problems have. That is that the models can only give a good accuracy classification on data that was trained on. This becomes a limitation already since our model was trained to distinguish 6 different classes, and if we want to launch our software with the ability to make it easy for photographers or consumers to sort their images, we are going to need to incorporate more classes. So, people can have a more complete solution.

The future work of this project would be to incorporate more classes into our data and or try to implement other algorithms such as transformers which we believe can have a positive result on our data.

Additions to our model we would like to make is incorporating ensemble models. Using ensemble models allows for a vote between multiple models. This has been shown to increase accuracy rates as it is relying on multiple trained models. If our original model struggled to identify a category over another maybe we could incorporate a second model trained only on two categories it struggles identifying and defer to that model allowing the model to cover up it's weaknesses.

Overall, the limits of our classifiers are that they are very time consuming. This is a big limitation since it requires a lot of time to train our models and if we want to fine tune our models or even make use methods like grid search it would take days to compute such methods.
]

9 YOUR FEEDBACK

[Overall, this was a great project idea. This final project allowed us to work with a large dataset of images and explore different data analysis on images, such as dimensional reduction, KMeans algorithms to find the most common color shade of images and getting the average shade of color, correlation on statistical operations. Steps that can be overlooked by the more attention grabbing convolutional neural networks.

This project also allowed us to work with large datasets and being able to deal with them. This was great because it allowed us to split our work into multiple steps that could be picked up without having to rerun already computed steps.

Additionally, working on this specific image multi-class classification problem allowed us to think how our project can be incorporated into real world applications and to which types of real-world issues could be aimed at. This extra bit of thought needed for the purpose of this project was great, given that most projects tend to lack a real-world application vision, and rather stay as just a learning activity.
]

10 REFERENCES

[Paper used for this project:

1. Yu, Q., Tang, H., Tan, K. C., & Li, H. (2013). Rapid feedforward computation by temporal encoding and learning with spiking neurons. *IEEE Transactions on Neural Networks and Learning Systems*, *24*(10), 1539–1552. https://doi.org/10.1109/tnnls.2013.2245677

2. Wen, Y., & Shi, P. (2007). Image PCA: A new approach for face recognition. *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*. https://doi.org/10.1109/icassp.2007.366139

3. Bhowmik, M. K., Bhattacharjee, D., Nasipuri, M., Basu, D. K., & Kundu, M. (2009). Classification of fused images using radial basis function neural network for human face recognition. *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*. https://doi.org/10.1109/nabic.2009.5393594

4. Georgiadis, P., Cavouras, D., Daskalakis, A., Sifaki, K., Malamas, M., Nikiforidis, G., & Solomou, E. (2007). PDA-based system with teleradiology and image analysis capabilities. *2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. https://doi.org/10.1109/iembs.2007.4352981

5. Hussain, M., Bird, J.J., Faria, D.R. (2019). A Study on CNN Transfer Learning for Image Classification. In: Lotfi, A., Bouchachia, H., Gegov, A., Langensiepen, C., McGinnity, M. (eds) Advances in Computational Intelligence Systems. UKCI 2018. Advances in Intelligent Systems and Computing, vol 840. Springer, Cham. https://doi.org/10.1007/978-3-319-97982-3_16

Links to the data used for this project:

Bansal, P. (2019, January 30). *Intel Image Classification*. Kaggle. https://www.kaggle.com/datasets/puneet6060/intel-image-classification

Links to relevant webpages used for this project:

Concyclics. (2022, April 6). *[0.995]dog 🐶 or cat 😾? VGG model introduction*. Kaggle. https://www.kaggle.com/code/concyclics/0-995-dog-or-cat-vgg-model-introduction

Team, K. (n.d.). *Keras documentation: Image Classification From Scratch*. https://keras.io/examples/vision/image_classification_from_scratch/

Team, K. (n.d.-a). *Keras Documentation: Efficientnetv2 B0 to B3 and S, M, L*. https://keras.io/api/applications/efficientnet_v2/#efficientnetv2l-function

Team, K. (n.d.-c). *Keras Documentation: Resnet and RESNETV2*. https://keras.io/api/applications/resnet/#resnet50v2-function

6. Zhang, H; Yu, T. Introduction to Deep Learning. In Deep Reinforcement Learning; Springer: Singapore, 2020.

7. Chollet, F. Getting Started with Neural Networks. In Deep Learning with Python; Manning: Shelter, Island, NY, 2018.

8. Shanmugamani, R. (n.d.). *Deep Learning for Computer Vision*. O'Reilly Online Learning. https://www.oreilly.com/library/view/deep-learning-for/9781788295628/1ab53400-64ab-43d7-9088-ad64943351f9.xhtml

9. Ai, S. E. (n.d.). What is EfficientNet? https://skyengine.ai/se/skyengine-blog/121-what-is-efficientnet#:~:text=EfficientNet%20is%20a%20convolutional%20neural,breadth%2C%20depth%2C%20and%20resolution.

10. Koehrsen, W. (2018, November 26). *Transfer learning with convolutional neural networks in Pytorch*. Medium. https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce

]