Freya Gray
CS3120
Final Project

1. **About The Project**

   For me, this project was about learning more about CNN, how to implement a CNN and image recognition in general. I am very interested in plants and I know that plant classification and disease detection by a human can be difficult. I use Google Lens for plant identification fairly often and I have always been interested in the technology that is used for plant identification. I used the Plant Village dataset which contains images of healthy and diseased plants which gave me a dataset I was interested in as well as a great dataset for training models and learning more about this aspect of machine learning. I did not set out to solve a problem but rather to learn about image recognition and CNN using a dataset that I am very interested in. I implemented three CNN models, two were trained by me while the other was pretrained.

2. **Sources and Work I Have Done**

   The AlexNet and ZFNet models are other people's ideas but implemented by me following the architecture described in each model's paper, the models were implemented in TensorFlow. The InceptionV3 model was implemented using TensorFlow's applications feature which has various pre-trained models for transfer learning. Guides on TensorFlow's website such as "Transfer Learning and Fine-tuning", "Convolutional Neural Network (CNN)" and "Image Classification" were used to learn how to load data, create a model, fit and train the model and how to implement a model for transfer learning. My code follows those guides' overall outline with some additions and alterations. "*Hands-on machine learning with Scikit-Learn, Keras and TensorFlow*" was also used to learn more about CNN, AlexNet and ZFNet and it gave me a starting point for my project. Both AlexNet and ZFNet use local response normalization which is not included as a Tensorflow layer, so I implemented it using a lambda layer. I also added a normalization layer for the AlexNet and ZFNet models (InceptionV3 requires normalized input). I tested the various parameters related to the Stochastic Gradient Descent optimizer as well as other parameters such as different layers and ways to implement a pretrained model. I added in the ability to save each model so the models can be tested fairly quickly and not require lengthy training times. I also trained each model on 10,25 and 50 epochs. The PlantVillage dataset was acquired from Kaggle. All sources are included at the end of the paper. Each python file cites the model and dataset used.

3. **Libraries and Packages used**
   a. matplotlib.pyplot
   b. tensorflow
   c. Tensorflow.keras: layers, Sequential, Model
   d. pathlib: Path

4. **Dataset**

   Data was acquired from Kaggle from the Plant Village Dataset. The data set contains images of healthy and diseased plants. The test set contains RGB, Grayscale and Segmented images, only the RGB images were used for training and testing. The dataset contains 54305 RGB images comprising 38 different classes of plants. The data was split using Tensorflow's image_dataset_from_directory function, where 80% was used for training and 20% for testing. The training set was split into 1358 batches of images with 32 images per batch. The test set was split into 340 batches, with 32 images per batch. All images were resized to 224 x 224 x 3 and all images were normalized before training.

5. **Models used**

Freya Gray
CS3120
Final Project

a. **AlexNet** - Developed in 2012 by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton, AlexNet is a CNN architecture that won the 2012 ImageNet challenge with an excellent error rate (Geron, 367). The model uses five convolution layers followed by three fully connected ones. The convolution layers and first two fully connected layers have a ReLU step. The first and second layers use local response normalization step before moving onto a max pooling step (Krizhevsky, 2-3). A summary of the architecture generated by TensorFlow is included below. I chose this model because it is a successful model for image recognition and is often referenced in papers about CNN and image recognition, both the other architectures reference AlexNet in their respective papers. It was also fairly easy to implement and gave me a good starting point for this project as well as a great way to learn about TensorFlow and how to implement Convolutional Neural Networks.

i. AlexNet Architecture

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 224, 224, 3)	0
conv2d (Conv2D)	(None, 56, 56, 96)	34944
lambda (Lambda)	(None, 56, 56, 96)	0
max_pooling2d (MaxPooling2D)	(None, 27, 27, 96)	0
conv2d_1 (Conv2D)	(None, 27, 27, 256)	614656
lambda_1 (Lambda)	(None, 27, 27, 256)	0
max_pooling2d_1 (MaxPooling2	(None, 13, 13, 256)	0
conv2d_2 (Conv2D)	(None, 13, 13, 384)	885120
conv2d_3 (Conv2D)	(None, 13, 13, 384)	1327488
conv2d_4 (Conv2D)	(None, 13, 13, 256)	884992
flatten (Flatten)	(None, 43264)	0
dense (Dense)	(None, 4096)	177213440
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16781312
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 38)	155686

Total params: 197,897,638
Trainable params: 197,897,638
Non-trainable params: 0

The first layer (called rescaling here) is a normalization layer to standardize pixel values between [0,1]
The first two convolution layers are followed by a lambda layer which invokes the local response normalization. The fully connected layers are the dense layers, the first two are followed by a dropout layer where .50 of the parameters are dropped out. The final layer uses a softmax function for the final prediction.

b. **ZFNet** - Developed in 2013 by Matthew Zeiler and Rob Fergus, ZFNet is a CNN architecture that improves upon AlexNet and won the 2014 ImageNet challenge (Geron, 368). It follows AlexNet's five convolution layers followed by three fully connected ones. The convolution layers and first two fully connected layers have a ReLU step. The first and second layers use local response normalization step before moving onto a max pooling step. However, the first convolution layer uses a 7x7 kernel and stride = 2 vs AlexNet's 11x11 kernel and stride = 4. The second convolution layer stride is increased from 1 to 2. The final three convolution layers are altered to increase the filters from AlexNet's (384,384,256) to ZFNets (512,1024,512) (Zeiler). The alterations to the final three convolution layers were not originally included but were a part of later tests that produced the best results for this model. I chose this model because it was an improvement of the AlexNet model, the changes were easy to implement and I learned how changing layers can alter a model.
    i. ZFNet Architecture

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 224, 224, 3)	0
conv2d (Conv2D)	(None, 112, 112, 96)	14208
lambda (Lambda)	(None, 112, 112, 96)	0
max_pooling2d (MaxPooling2D)	(None, 55, 55, 96)	0
conv2d_1 (Conv2D)	(None, 28, 28, 256)	614656
lambda_1 (Lambda)	(None, 28, 28, 256)	0
max_pooling2d_1 (MaxPooling2	(None, 13, 13, 256)	0
conv2d_2 (Conv2D)	(None, 13, 13, 512)	1180160
conv2d_3 (Conv2D)	(None, 13, 13, 1024)	4719616
conv2d_4 (Conv2D)	(None, 13, 13, 512)	4719104
flatten (Flatten)	(None, 86528)	0
dense (Dense)	(None, 4096)	354422784
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16781312
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 38)	155686

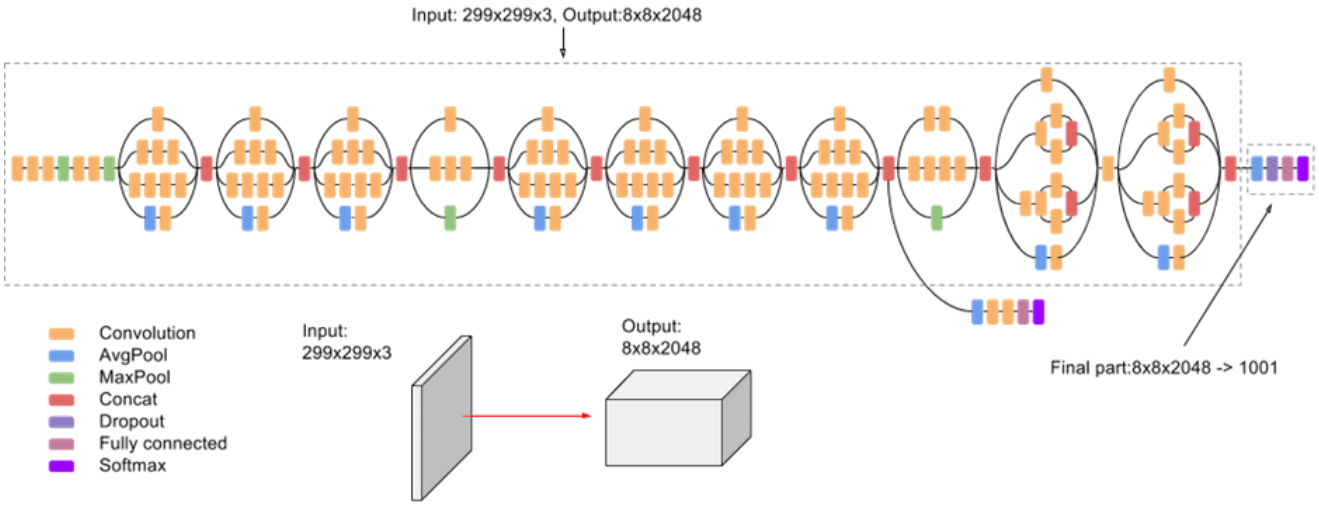
Total params: 382,607,526
Trainable params: 382,607,526
Non-trainable params: 0

Most of this architecture is the same as AlexNet. The first two convolution layers have a larger output size and third, fourth and fifth convolution layers have more parameters.

c. **InceptionV3-** The third version of Google's Inception net, introduced in 2015. This model has been trained on the ImageNet dataset where it has achieved over 78.1% accuracy(Google). This model is much more complex than the previous two and represents a current way to classify images. One of the main ideas behind the model is the individual inception modules. Rather than having a large convolution layer, the model splits convolution layers into multiple smaller ones. The Inception module improves the overall model efficiency (Krizhevsky). The graph below illustrates the overall model architecture.
I chose this model as it was included in Tensorflow's pretrained models as well as it is a widely used model that has been shown to achieve high accuracy. It also

gave me a way to learn more about transfer learning and how to apply a pre trained model.

i. Inception V3 Overall architecture



(Google)

ii. Inception V3 Transfer Learning Architecture

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 224, 224, 3)]	0
tf.math.truediv (TFOpLambda)	(None, 224, 224, 3)	0
tf.math.subtract (TFOpLambda	(None, 224, 224, 3)	0
inception_v3 (Functional)	(None, 5, 5, 2048)	21802784
global_average_pooling2d (Gl	(None, 2048)	0
dropout (Dropout)	(None, 2048)	0
dense (Dense)	(None, 38)	77862

Total params: 21,880,646
Trainable params: 77,862
Non-trainable params: 21,802,784

The first layer sets the input size of an image. The next two layers normalizes pixel values between [-1,1]. Next layer is the Inception V3 model with all training layers frozen. In the first graph of the InceptionV3 model only the part of the model labeled final part is not frozen, meaning the weights previously trained on the ImageNet dataset are preserved.The global average pooling layer calculates the average of each feature map in the previous layer, which is followed by a dropout layer where 20% is

dropped out. Finally the last layer uses a softmax function for final prediction.

6. **Analysis**

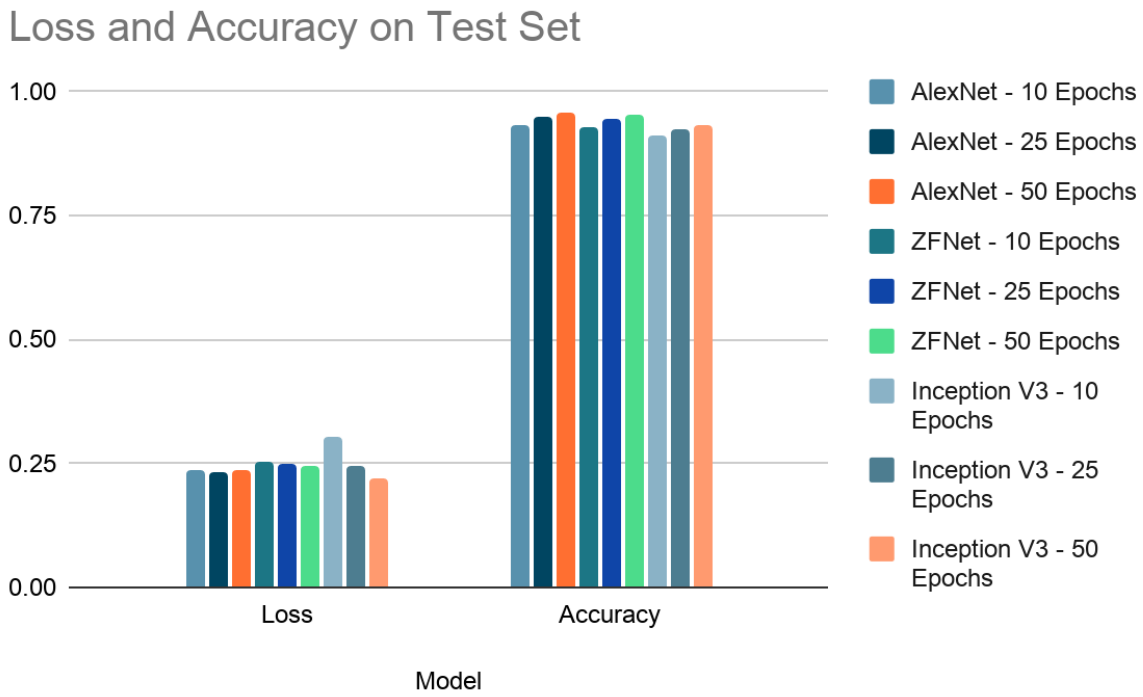
When I started this project I was focussed on implementing just the AlexNet model and I went through various hyperparameters. On the first run of my model, I did not use a normalization layer and used Stochastic Gradient Descent with a learning rate of 0.00001. After training for 50 epochs, the model produced a loss of .1812, accuracy of .9409 and a validation loss of .1687 and validation accuracy of .9489. Initial implementation of ZFNet with the same hyperparameters produced similar scores with a loss of .1247 accuracy of .9686, validation loss of .1343 and validation accuracy of .9588. The ZFNet model was able to reach higher accuracy in less epochs than AlexNet.

Next I added in a normalization layer and found that with a learning rate of 0.00001, the model took an extremely long time to train and produced low accuracy and high loss. Switching to a learning rate of 0.001 and using Nestorov movement greatly improved training time, accuracy and validation. However, with a smaller learning rate and a no normalization layer, both AlexNet and ZFNet produced higher accuracy and lower loss.

While implementing the Inception V3 model, I initially had very low accuracy and high loss, this was due to not all the training layers being frozen. After ensuring the training layers were frozen, I tested various layers to add to the pretrained model. I found that the global average pooling layer followed by a dropout layer produced the best results but others such as adding in other dense layers of various sizes still produced fairly accurate results, around 90%.

Finally, I tested all the models using the models I had saved from training. Each model had a saved model trained using 10,25 and 50 epochs. They all used the Stochastic Gradient Descent optimizer, learning rate = 0.01, movement = .9 and using Nesterov Momentum. Loss was calculated with categorical cross entropy. And all models used a normalization layer. The graph below shows that overall, AlexNet performed the best over all epochs.

Freya Gray
CS3120
Final Project



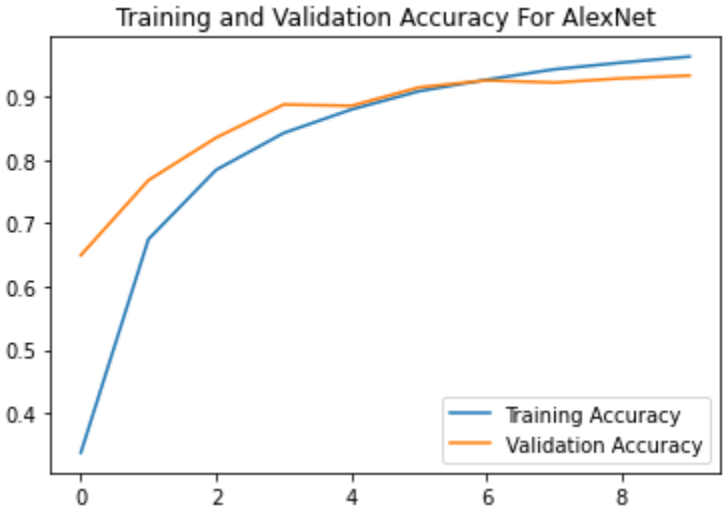
Loss and Accuracy on Test Set

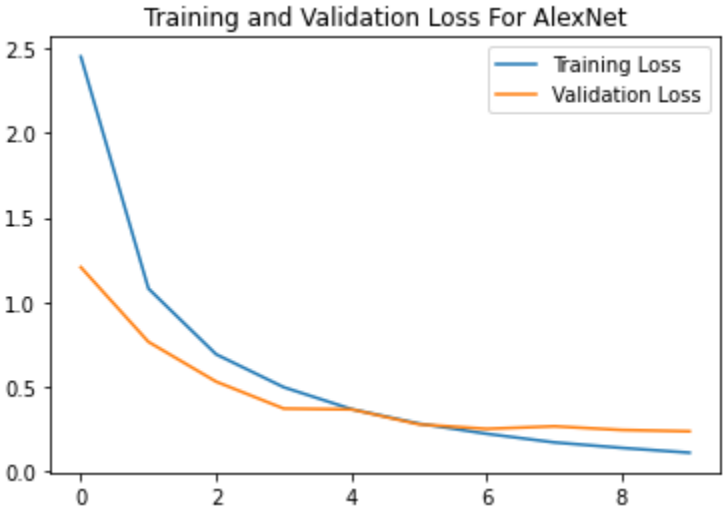
Model	Loss	Accuracy
AlexNet - 10 Epochs	0.23713259398937225	0.933063268661499
AlexNet - 25 Epochs	0.23322737216949463	0.9494521617889404
AlexNet - 50 Epochs	0.23610271513462067	0.9580149054527283
ZFNet - 10 Epochs	0.2521952688694	0.9299327731132507
ZFNet - 25 Epochs	0.24878114461898804	0.9437436461448669
ZFNet - 50 Epochs	0.24605795741081238	0.9523063898086548
Inception V3 - 10 Epochs	0.30422258377075195	0.9122548699378967
Inception V3 - 25 Epochs	0.2463376224040985	0.9250529408454895
Inception V3 - 50 Epochs	0.2180255651473999	0.9312217831611633

Freya Gray
CS3120
Final Project

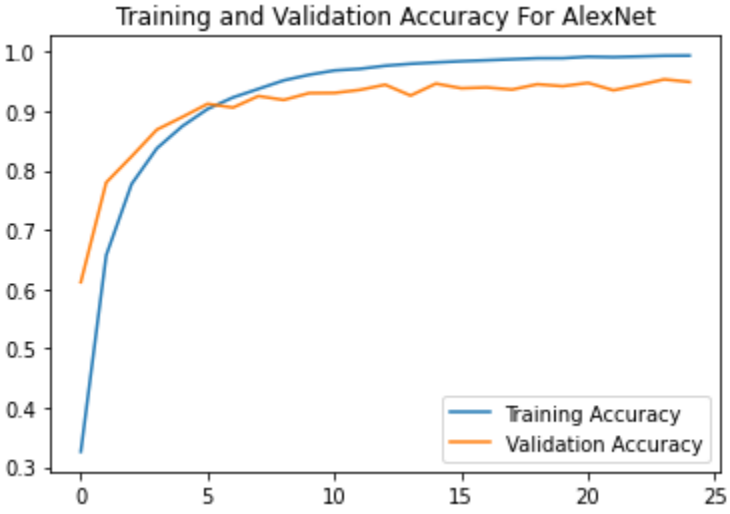
The following graphs were generated with the Stochastic Gradient Descent optimizer, learning rate = 0.01, movement = .9 and using Nesterov Momentum. Loss was calculated with categorical cross entropy. All graphs and test numbers were generated with all models using a normalization layer.

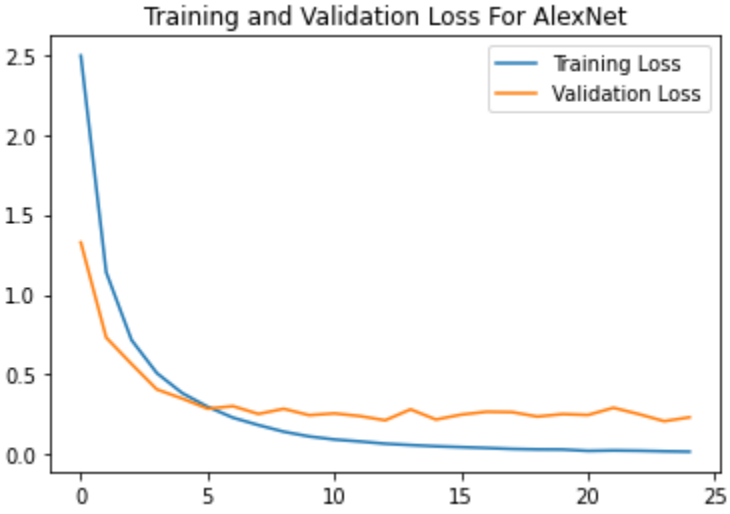
a. AlexNet Accuracy and Loss Graphs for 10,25 and 50 epochs

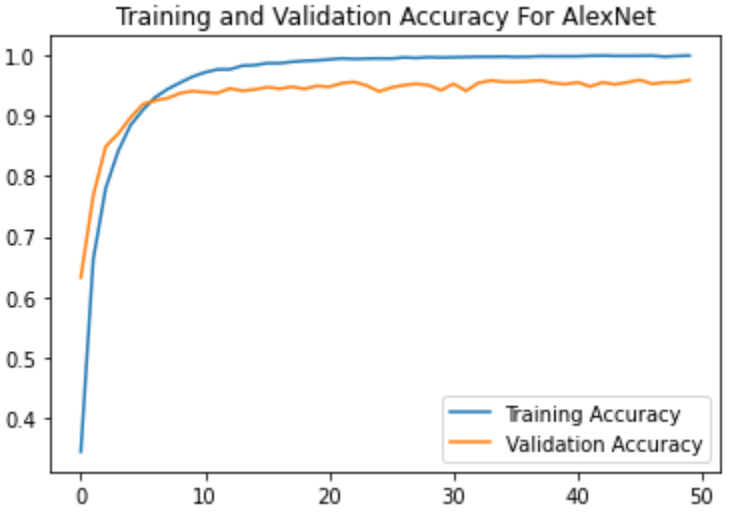




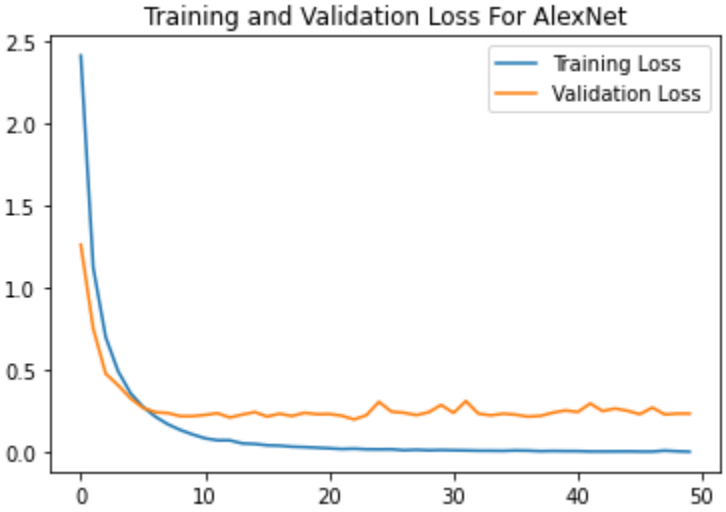
Freya Gray
CS3120
Final Project



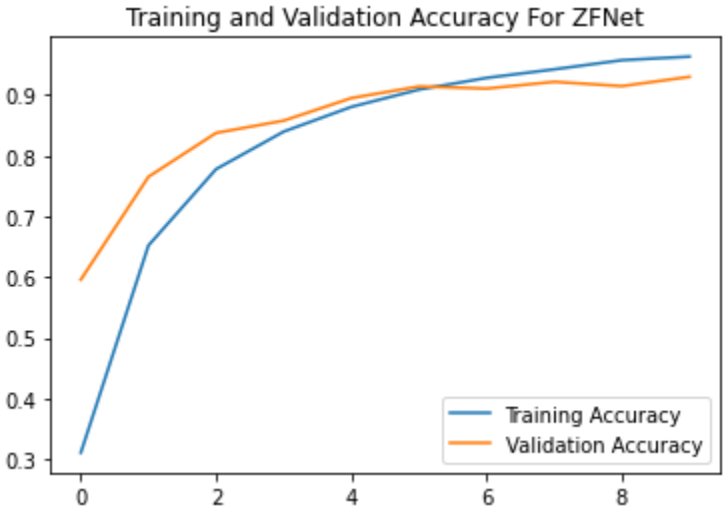


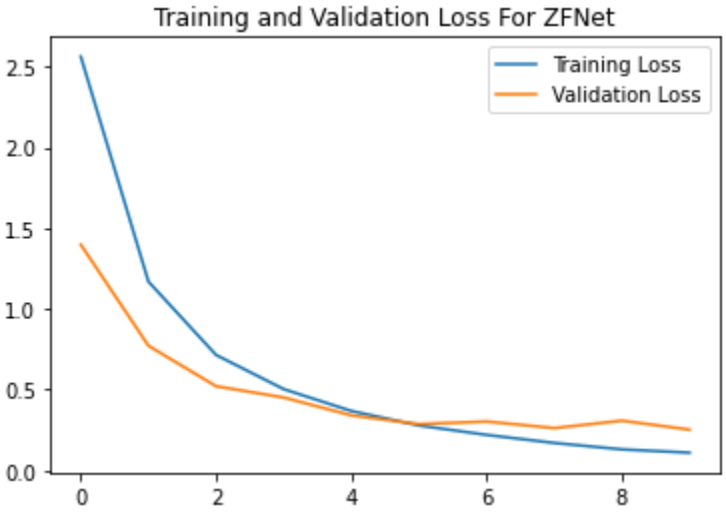


Freya Gray
CS3120
Final Project

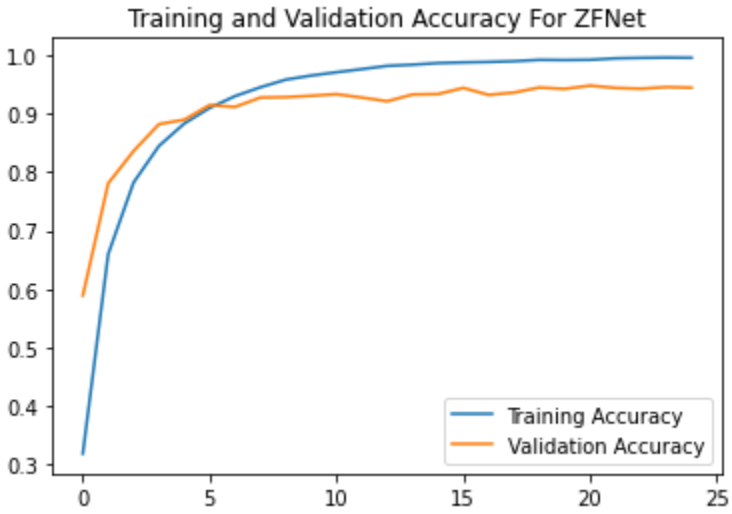


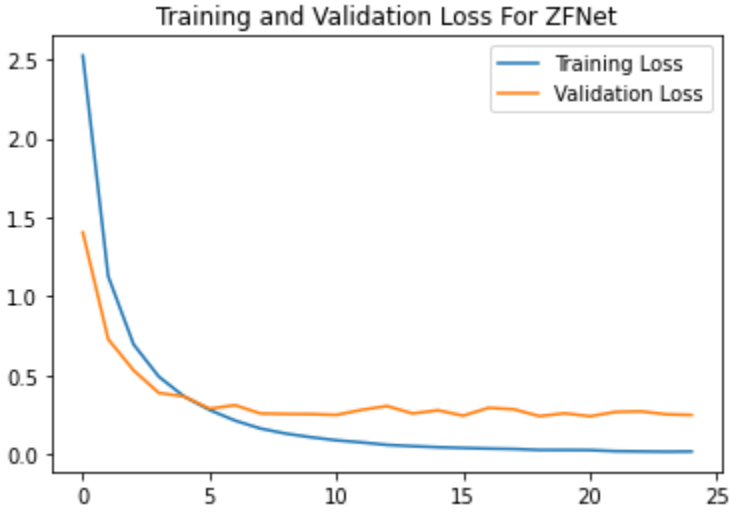
b. ZFNet Accuracy and Loss Graphs for 10,25 and 50 epochs



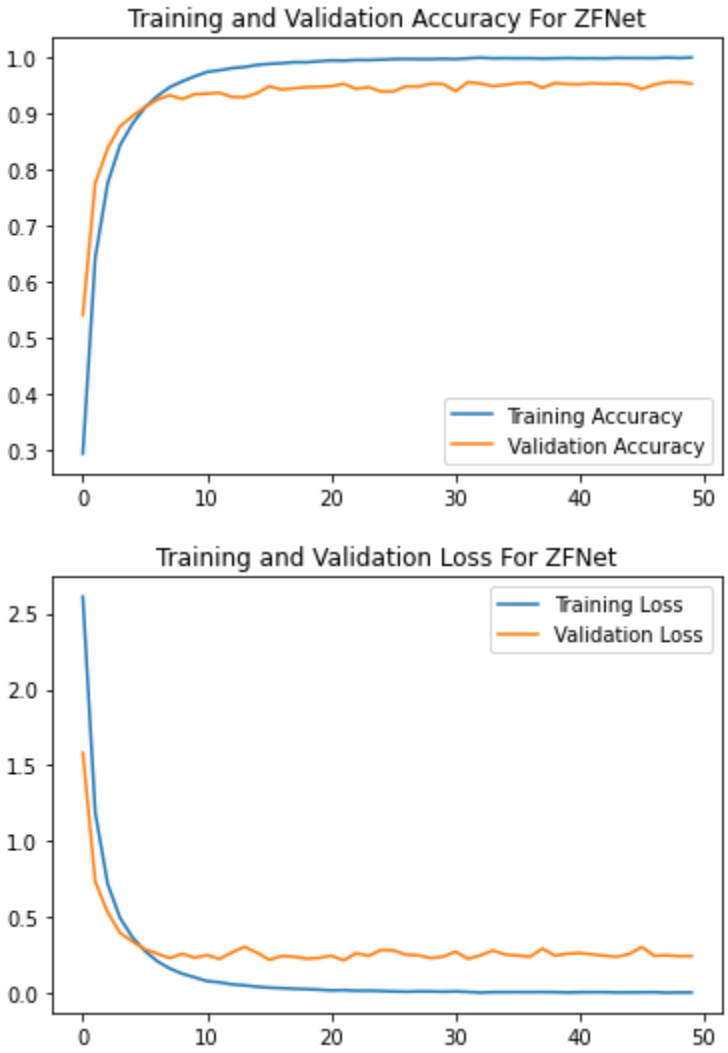


Freya Gray
CS3120
Final Project



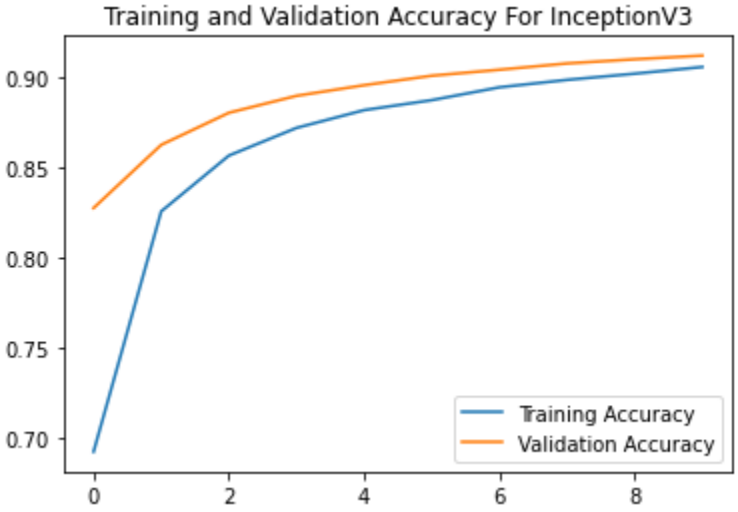


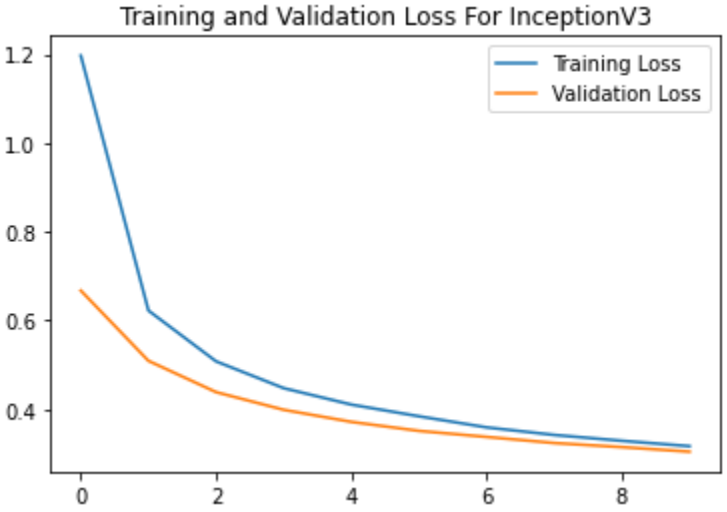
Freya Gray
CS3120
Final Project

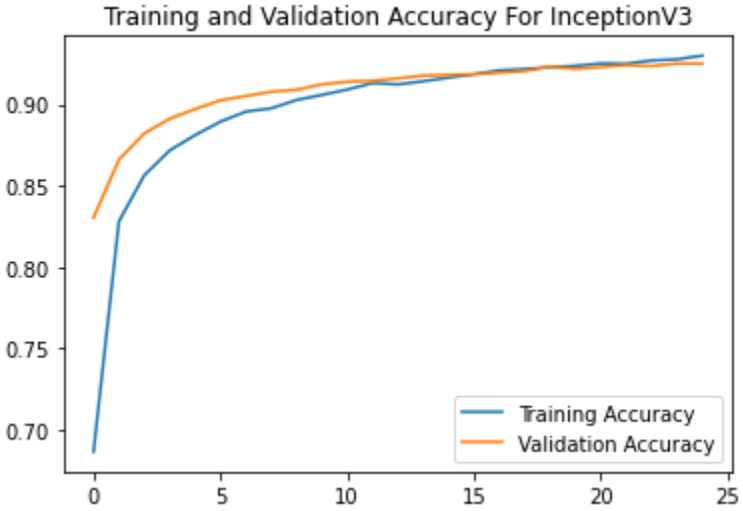


c. InceptionNet Accuracy and Loss Graphs for 10,25 and 50 epochs

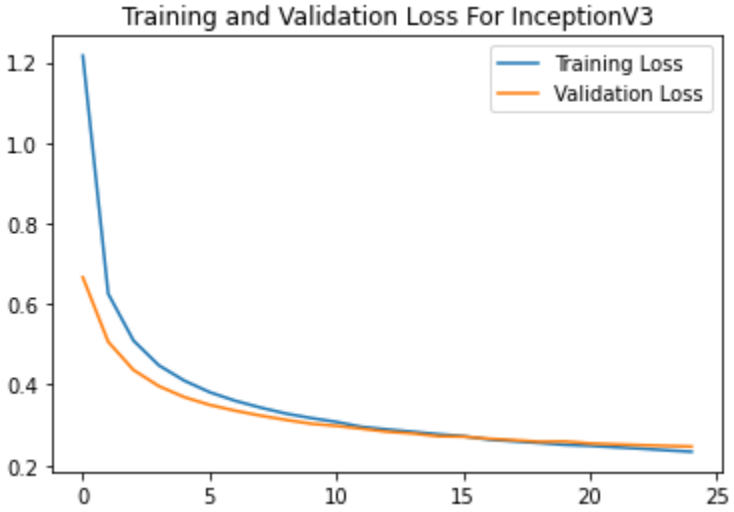
Freya Gray
CS3120
Final Project

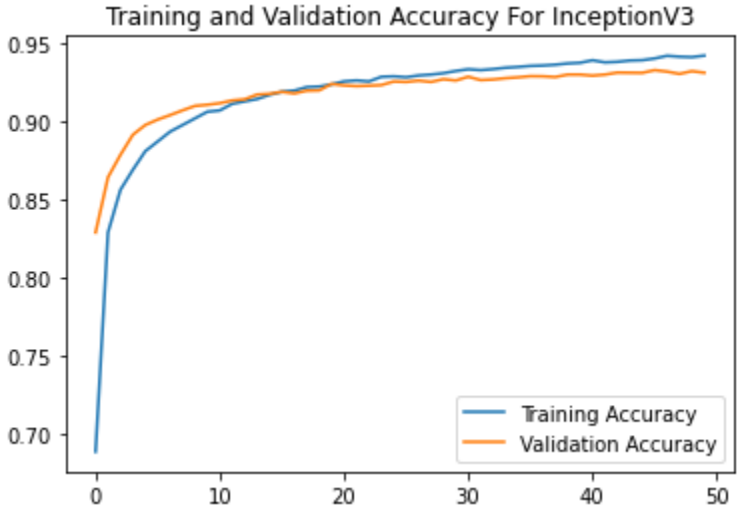


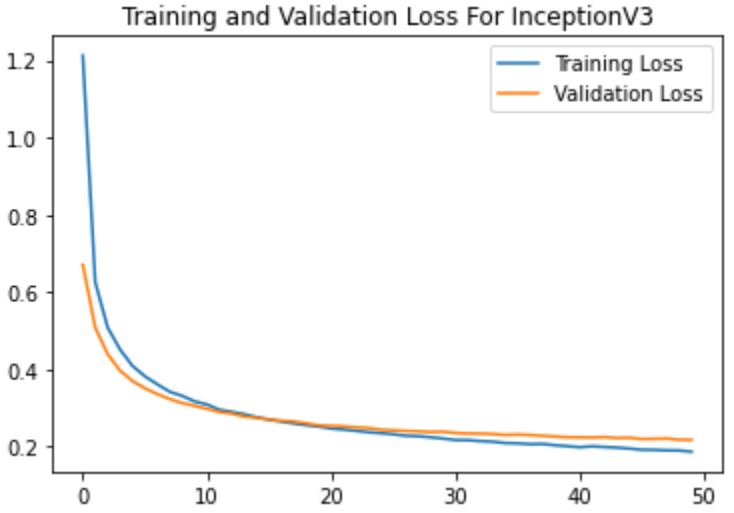




Freya Gray
CS3120
Final Project







Freya Gray
CS3120
Final Project

Sources

"Advanced Guide to Inception v3 on Cloud TPU " Google, cloud.google.com/tpu/docs/inception-v3-advanced.

Convolutional Neural Network (CNN) TensorFlow Tutorial. TensorFlow. https://www.tensorflow.org/tutorials/images/cnn.

Géron Aurélien. (2019). *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O'Reilly.

Image classification : TensorFlow Tutorial. TensorFlow. https://www.tensorflow.org/tutorials/images/classification.

Load images : TensorFlow Core. TensorFlow. https://www.tensorflow.org/tutorials/load_data/images.

Transfer learning and fine-tuning. TensorFlow. https://www.tensorflow.org/tutorials/images/transfer_learning.

Dataset: PlantVillage
Hughes, D. P. & Salathé, M. *PlantVillage Dataset - Images of Healthy and Diseased plants.* https://www.kaggle.com/abdallahalidev/plantvillage-dataset

Model: AlexNet
Krizhevsky, A., Sutskever, I., & Hinton, G.E. (2012). *ImageNet classification with deep convolutional neural networks.* Communications of the ACM, 60, 84 - 90.

Model: InceptionV3
Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, & Zbigniew Wojna. (2015). *Rethinking the Inception Architecture for Computer Vision.*

Model: ZFNet
Matthew D Zeiler, & Rob Fergus. (2013). *Visualizing and Understanding Convolutional Networks*.