# Classifying Military Organizations with Deep Learning

Tony Urschler

May 2022

**Abstract**

Due to its powerful nature and flexibility, you'd be hard-pressed to find an industry that is not at least exploring how deep learning could impact their technology and goals. This is no different for defense organizations all around the globe. One issue that every individual combat soldier will experience is determining whether a given target is a friend or foe. Uniformed militaries can have very similar uniform patterns, which can cause confusion on the battlefield. This confusion undoubtedly leads to unnecessary loss of life, not only through missing targets of opportunity, but also because of fratricide.

The goal of this project is to determine whether a viable deep learning model can be created to classify images based on the military organization within. Since this project is more theoretical, there are only three organizations that are being classified. They are the United States Army, the Israeli Defense Forces, and the Chinese People's Liberation Army Navy Marine Corps. In this report, we focus on the process of achieving this goal. This includes everything from data collection and normalization to the usage of recent architectures and transfer learning. Through this exploration, we provide information on the best strategies for attempting to tackle this problem, as well as well as insight on why certain methods don't work. In the end, the final model created achieves more than a 96% test accuracy and a test loss of 0.19. These scores show promise in developing systems that can aid military members in target identification, and this report should provide others with a good starting point to implement these systems.

## 1 Introduction

The purpose of this project is to create a model, using Deep Learning, that will be able to categorize images according to the military personnel within. This is a multi-class single-label problem since there are three categories. They are the People's Liberation Army Navy Marine Corps (PLANMC), the United States Army (USA), and the Israeli Defense Force (IDF). These three military organizations were chosen based on their different uniforms. The United States Army and the Israeli Defense Force have similarly colored uniforms, while the People's Liberation Army Navy Marine Corps are quite different.

## 2 Data Preparation

### 2.1 Collection

The data collection was done using Google Image Searches. These searches used key words to maximize the number of qualifying images on the page. Qualifying images are considered diverse (having various amounts of personnel and scenery), and unique (have not been added to the data set yet). To speed up the process of downloading the individual images, the Image Downloader Chrome Extension was used to easily scrape images from the web page.

### 2.2 Preprocessing

The images were filtered by hand to ensure the data set did not contain duplicates, irrelevant images, or images that were of poor quality. All the images were then cropped to a size of 256x256 to ensure all images had the correct aspect ratio. This was done by using a mass cropping tool which can be found at

https://www.birme.net/. This tool allows the user to adjust the area being cropped on all images prior to execution, which ensured that each image was cropped properly.

## 2.3 Visualization

As seen in Figure 1, the PLANMC have blue, grey, and white uniforms, which makes this organization much more unique that the other two. While the IDF doesn't have a pattern, the green used in their uniforms is very similar to the color used in the USA's. The images may contain one or multiple uniformed members, as well as non-uniformed individuals. The backgrounds of the images are also as diverse as possible.


Figure 1: Images from different categories of the data set

## 2.4 Distribution

The distribution, by and large, is equal, though there are approximately 50 more images for both the USA and the IDF, which can be seen in Figure 2. This was done purposefully, since it is assumed that categorizing these two organizations will be more difficult. This will give the model a better chance at detecting the differences between the two organizations.

## 2.5 Normalization

To normalize the data, the ImageDataGenerator from Keras will be used to re-scale each image with a 1/255 ratio. Since these are all colored images, this will allow all three channels of each image to be efficiently normalized.

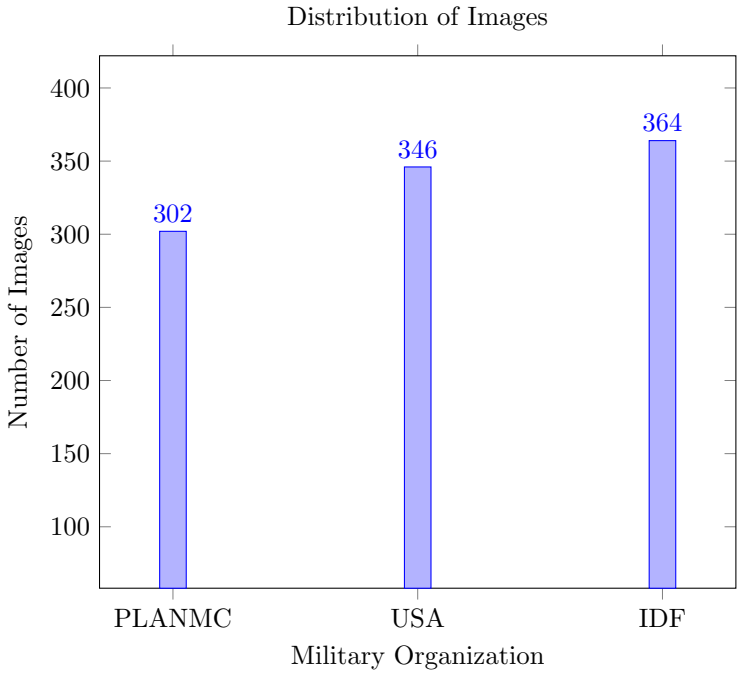


Figure 2: Bar Graph showing the distribution of images for the PLANMC, USA, and IDF image data set

# 3 Over-training a Model

## 3.1 Design Decisions

In order to create a small Convolutional Neural Network that overfits the data set, some criteria must be kept in mind in the designing phase. Since the neural network is meant gain most of its knowledge from the convolutional layers, it is important that there aren't a large number of parameters in the last dense layers. This would essentially make the model a Feed-Forward NN, which is not the intention. The goal is also to create the smallest network possible, which will be considered anything larger than 15,000 parameters.

With these criteria in mind, it is now time to develop a general design for the model. Through brute force, it was systematically determined that utilizing Batch Normalization periodically made the data more stable as it is flowing through the model, making it train more efficiently. Max Pooling was also used in strategic locations to ensure the amount of parameters didn't get out of hand. In the end, this strategy created a model that was trainable, condense, and had the majority of it's parameters in convolutional layers.

## 3.2 Analysis of Designs

Four models were created and tested with the aforementioned criteria to see which combination of layers and filters is the most efficient. Each of their change in accuracy, precision, recall, and loss can be seen in Figure 3. Looking at the first row of charts, it is apparent that they did not have the capacity to overfit on the dataset. The top-left has only four convolutional layers, while the top-right has five. It can be seen that their learning, despite this difference, is relatively the same in the end. Both only achieve an accuracy of about 80%, and both still have relatively high losses.

The graph in the bottom-left corner has the exact same design as the model on the top-right, but with three additional filters (two additional in one layer, and one additional in another). The result of this model far surpasses that of the previous. After approximately 80 epochs, the model nearly reaches 100% accuracy, precision, and recall. It also brings its loss down to near zero.

The final chart depicts a model that is the exact same as the bottom-left, but has filters of size 4x4 rather than 3x3. This increase in filter size nearly doubled the amount of parameters, which made the training of

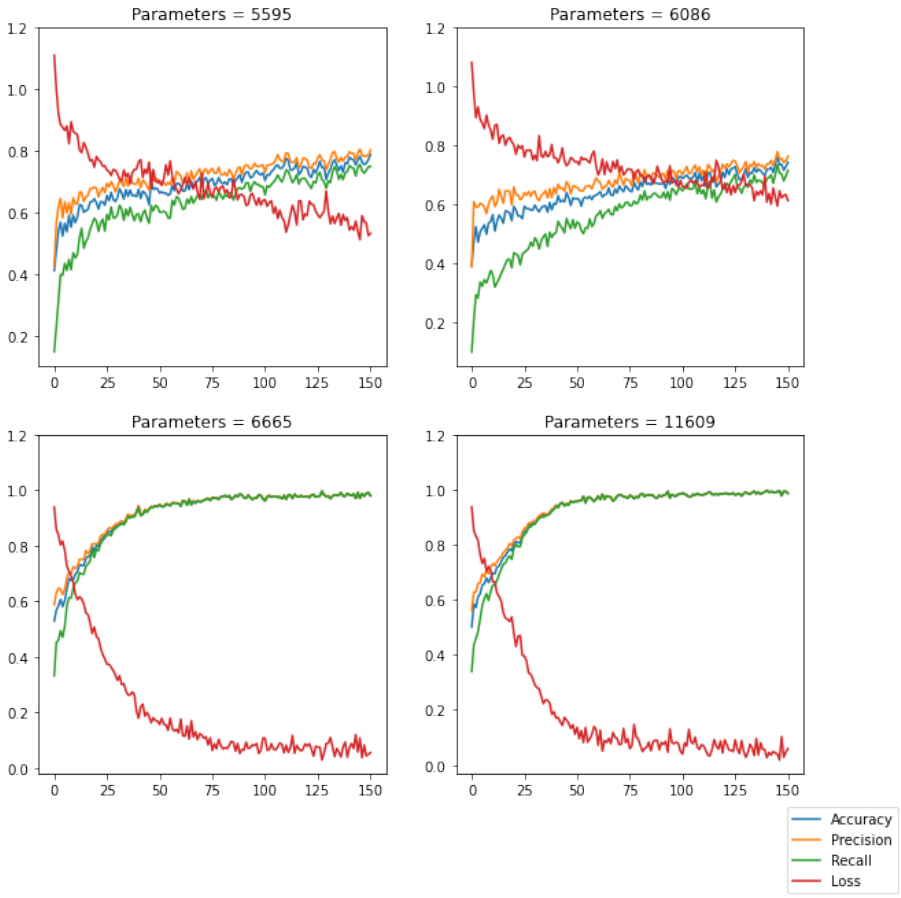


Figure 3: Learning curves for the overfit attempts

this model much more efficient (as it finished in nearly 100 epochs). But in terms of achieving near 100% accuracy, it did so in nearly the same amount of epochs as the previous model.

### 3.3 Choosing the Model

The model displayed in the bottom left, was chosen out of the four as the model used to overfit. This model was chosen because it had the highest accuracy, precision, and recall, which was 99.51%, and the lowest loss at 0.0253. It also appears to be in the "sweet spot", where anything with fewer parameters cannot overfit the data set, and anything with more accomplishes the same task, just in a more efficient manner.

## 4 Using Validation and Test Sets

Now that a convolutional neural network architecture that overfits is known, the next step is to begin generalizing the model with the use of validation and test sets. The purpose of a validation set is to aid the model in achieving generality. And the test set is used as an unbiased data set to see how the model truly performs.

The validation set was randomly selected from the overall data set, and amounts to 20% (or 201 images) of the total images. The test set was also randomly selected and it encompasses 20% of the overall data set. In order to achieve the highest validation set accuracy, the current architecture will be manipulated until this goal is met.

| Model | CNN Layers | M.P. | B.N. | Parameters | Valid Acc. | Test Acc. |
|---|---|---|---|---|---|---|
| OvrFit | 5 | 3 | 1 | 6,665 | 0.73 | 0.69 |
| Rev. 1 | 5 | 3 | 2 | 33,669 | 0.70 | 0.64 |
| Rev. 2 | 6 | 3 | 5 | 46,603 | 0.80 | 0.75 |
| Rev. 3 | 8 | 3 | 7 | 65,355 | 0.79 | 0.62 |

Figure 4: Overviews of Various Models' Architecture. M.P. refers to MaxPool2D Layers and B.N. refers to BatchNormalization Layers.

### 4.1 Using Validation Data to Train the Overfitted Model

The best over-fitted model is trained first, whose architecture can be found in Figure 4. After training, the validation accuracy was well above the baseline at 73%, and also performed similarly well on the test set with an accuracy of 69%. But, as the graph of the Overfit Model shows in Figure 5, the validation accuracy does not follow the training accuracy in growth, and the validation loss is incredibly high. In order to have the validation accuracy better mirror the training accuracy, it was determined that increasing the models size would allow this to happen.

### 4.2 Architecture Revision #1

The first revision performed worse than the overfit model, with a validation accuracy of only 70%. It also obtained an even lower test accuracy of only 64%. Even though the validation accuracy is better, the "Revised Model #1" plot in Figure 5 shows that the validation and training accuracies remain distant while the validation loss has gotten worse. It was decided to increase the size of the model, as well as use more Batch Normalization layers to hopefully bring the training and validation accuracies closer together.

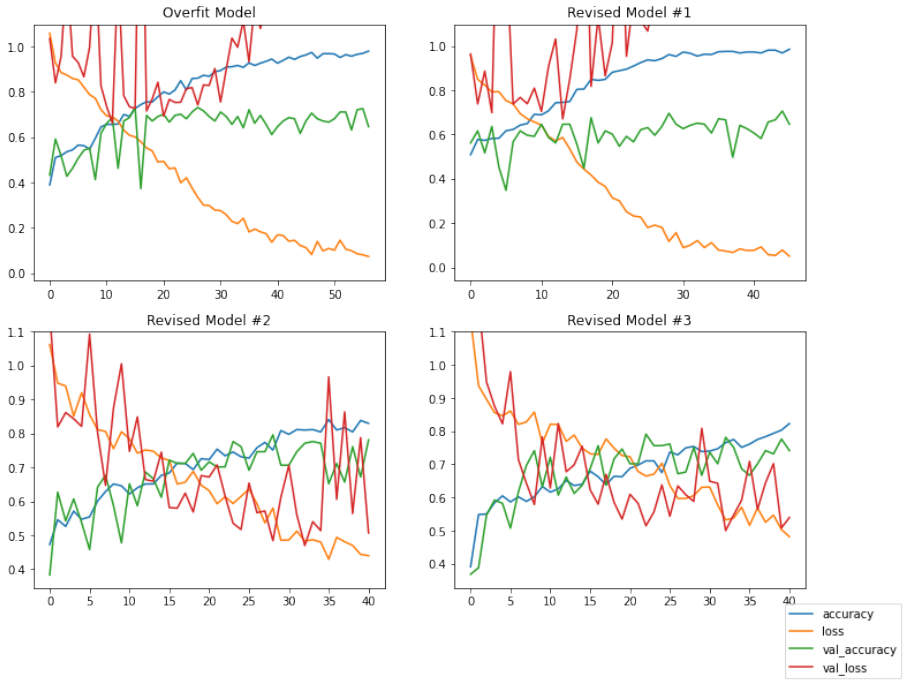


Figure 5: Various Architectures Used

### 4.3 Architecture Revision #2

The second architecture revision proved to be a success. With a validation accuracy of 80% and a test accuracy of 75%, it proves to be better than the previous models. The validation and training accuracies mimic one another and even the validation loss trend is similar to the training loss. In an attempt to continue improvement, the same methodology was used to produce the next revision. The size was increasing, as well as the use of more Batch Normalization layers.

### 4.4 Architecture Revision #3

This revision performed worse than the last with only 79% validation accuracy and 62% test accuracy. This model acts more erratic than the previous one, as shown in Figure 5. While the training accuracy increases at a nice pace, the validation accuracy dances around. We also see the same thing happen with the training and validation losses. This instability is what likely caused the validation accuracy and test accuracy to be so wildly different in this case.

### 4.5 Choosing an Architecture

In regards to validation and test accuracy, Architecture Revision #2 is obviously the best model created of the four. Not only does it score the highest in these two metrics, but it mirrors the training accuracy and loss very well.

## 5 Using Augmented ImageDataGenerators

Now that a preferred architecture has been determined, the aim becomes to better generalize the model by achieving better validation and test accuracies. In this section we will experiment with augmented ImageDataGenerators to determine which combination of parameters helps us come closer to achieving this goal. All models will be trained for approximately 200 epochs so we can gain a better understanding of how their learning curves compare.

### 5.1 No Augmentation

To create a baseline, the first ImageDataGenerator will apply no augmentation to the images, just as it has done in the previous sections. This method performed fairly well, achieving 79% for validation accuracy and 72% on the test accuracy, as shown in Figure 7. But, as is depicted in Figure 6, the validation accuracy and loss do not mirror the training accuracy well. This indicates it has done a poor job at generalizing, and the model begins to overfit on the training set after around 25 epochs.

### 5.2 Rotation Augmentation

Adding a rotation range of 35 degrees shows improvement in both validation and test accuracies. They have increased to 85% and 77% respectively, but the true testament of the increased performance this generator brought is shown in Figure 6. The training and validation accuracy and loss have the same general trend, and they only start to break apart around epoch 150. This indicates that a form of generalization has begun to occur.

### 5.3 Width and Height Shift Augmentations

Swapping out the rotation range augmentation for width and height shift augmentations of 0.2, there is another increase in performance. The validation accuracy is up to 89% and the test accuracy is back to 83%. These scores are not the only an improvement. We can also see in Figure 6 that this augmentation, Augmentation #2, has much better accuracy and loss curves. Both the validation accuracy and loss still follow the training.

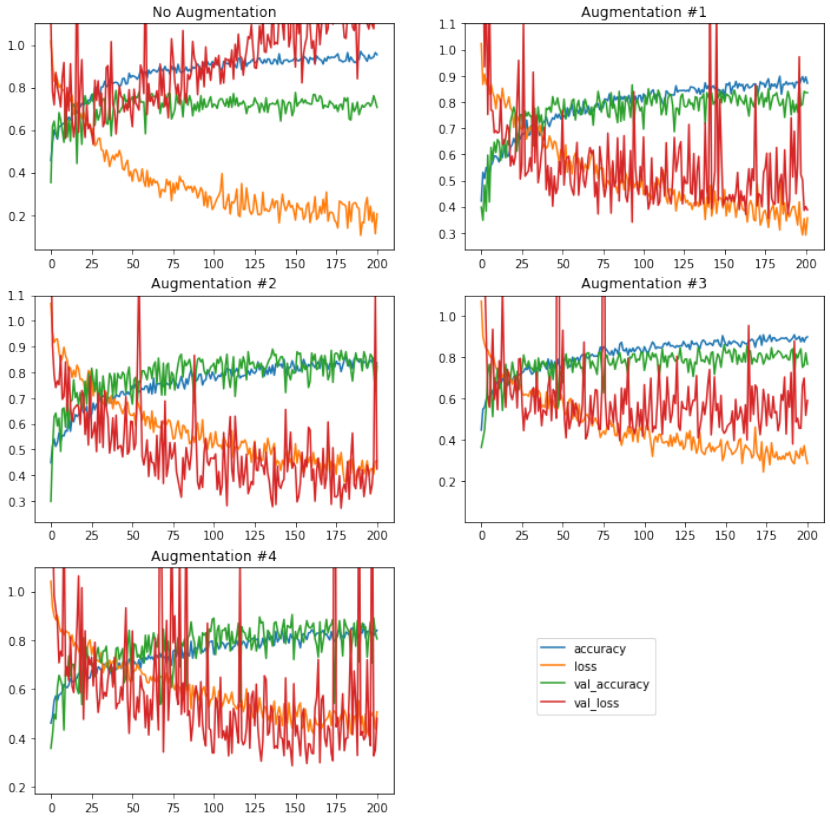


Figure 6: Various Augmented ImageDataGenerators Learning Curves

### 5.4 Sheer and Zoom Augmentations

Using the sheer and zoom range augmentations of 0.2 show a decrease in performance when compared to the previous augmentation, with only a validation accuracy of 86% and test accuracy of 80%. And while the validation curves of Augmentation #3 in Figure 6 follow the training curves better than having no augmentation, it breaks away around epoch 75. It is obvious that it performs worse than the width and height augmentation.

### 5.5 Combining all Augmentations

Combining all the previous augmentations into one generator proved to be the most effective. With a validation accuracy of 90% and a test accuracy of 80%, it has some of the best scores out of all the previous ImageDataGenerators. It also follows the training curves the best as shown in Figure 6. Both the validation accuracy and loss have lower variability, which can be seen by both their curves not jumping as wildly as the previous. They also do not deviate from the training curves, showing the model has achieved some kind of generality.

## 6 Using Regularization Techniques

Now that an optimal ImageDataGenerator has been found, it is time to experiment with different regularization techniques. The goal is to leverage the use of regularization to achieve higher validation accuracies, indicating more generalized models. The models will be trained with their early-stopping callbacks set to 200 epochs so we can gain a better understanding of how their learning curves compare.

| Generator | Description | Valid Acc. | Test Acc. |
|---|---|---|---|
| Regular | No augmentation | 0.79 | 0.72 |
| Gen. 1 | Rotation range: 35 | 0.85 | 0.77 |
| Gen. 2 | Width & height shift: 0.2 | 0.89 | 0.83 |
| Gen. 3 | Sheer & zoom range: 0.2 | 0.86 | 0.80 |
| Gen. 4 | All previous | 0.90 | 0.80 |

Figure 7: Overview of ImageDataGenerators and their corresponding accuracies.

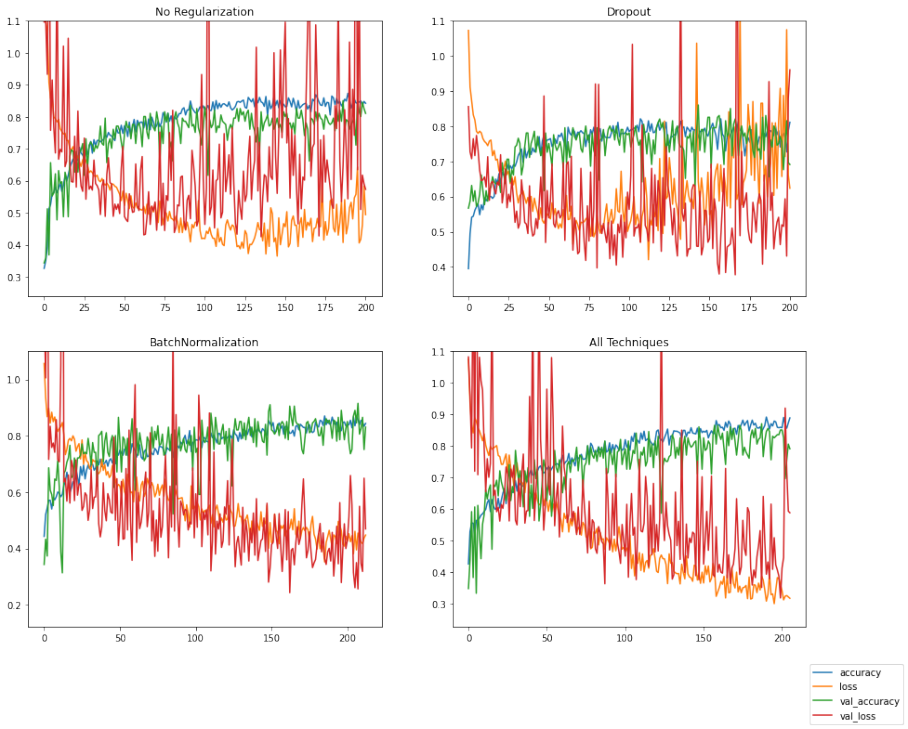


Figure 8: Learning curves of models with various regularization techniques

## 6.1 No Regularization

Using the model without any regularization techniques performs very poorly. As shown in Figure 9, it does have a decent validation accuracy of 84%, but it's test accuracy and test loss are very poor at just 76% and 1.32 respectively. We can also see in the "No Regularization" chart in Figure 8 that the validation and training loss never dip below 0.45, and the validation loss' variability is incredibly high.

## 6.2 Dropout Layers

After introducing Dropout layers, the model's variability begins to decrease as shown in the "Dropout" chart, but it's losses still never drop below 0.4. It's accuracy scores just barely improve, with validation and test accuracies being 85% and 78%. That's just a 1 and 2% increase from no regularization. The test loss on the other hand has largely decreased, coming down to 0.64. This model definitely performs better than having no regularization, but it is far from perfect.

| Model | Valid Acc. | Test Acc. | Valid Loss | Test Loss |
|---|---|---|---|---|
| No Regularization | 0.84 | 0.76 | 0.48 | 1.32 |
| Dropout | 0.86 | 0.75 | 0.46 | 0.64 |
| BatchNormalization | 0.91 | 0.89 | 0.33 | 0.31 |
| Both | 0.87 | 0.78 | 0.35 | 0.63 |

Figure 9: Overview of regularization techniques used and their corresponding accuracies.

### 6.3 BatchNormalization Layers

The introduction of BatchNormalization layers increases the model's performance substantially. The validation and test accuracy increase to 91% and 89% respectively. On top of this the validation and test loss also decrease to 0.33 and 0.31. These scores are much better than the utilization of Dropout layers, let alone no regularization at all. The "BatchNormalization" chart shows a much better performance as well, since the losses continue to fall with low variability, and the accuracies grow well with one another. This is also the technique that has been employed in the previous sections, and so far this proves it was the correct decision.

### 6.4 Combining Techniques

When combining Dropout and BatchNormalization layers, it can be seen that the model's performance begins to drop, just by looking at the "All Techniques" chart. We can see the validation loss more poorly reflects the training loss, and the validation accuracy begins to drop away from the training accuracy, which indicates overfitting is beginning to occur. But once you look at figure 9 it is obvious it performs much worse. It's validation accuracy is 87%, but the test accuracy falls to 78%. It's test loss also increased to 0.63.

### 6.5 Choosing the Regularization Technique

It is quite obvious that the best performing model uses just BatchNormalization layers. This method not only has superior metrics compared to it's tested counterparts, it shows a greater capability to generalize as shown by it's graph. Both the validation accuracy and loss mirror their training counterparts very well, with fairly low variability. This explains why it also has the highest test accuracy, as it is more capable of learning the intricacies of the different categories, rather than just memorizing some pieces of images.

## 7 Experimenting with Residual Neural Networks

Residual Neural Networks assist in reducing the amount of information that is lost through the convolutional, pooling, and dropout layers. This is done by adding the input vector back to the manipulated vector after going through the layers. The reduction of information loss should allow a residual network to achieve higher accuracy, and thus be more powerful. In this section, experiments were conducted to test what configuration of a residual neural network will provide the best validation and test scores on the dataset.

### 7.1 Analyzing the Residual Architectures

Architecture #1 has the best performance of the four different models. As shown in Figure 11, its test accuracy and loss have the best balance of the four models, having scored 82% and 0.55 respectively. While this model does begin to overfit around epoch 80, as shown in Figure 10, it has far less variability in validation accuracy and loss up until that point. Meaning that during that time, it was achieving some form of generalization. Other models may not have overfit at all, like Architecture #3, or overfit at a later epoch, like Architecture #4, but they did not achieve as good of scores as #3. They also have a higher variability in their scores from epoch to epoch.

All in all, the residual neural network does provide a more consistent set of test scores than the previously tried approaches. In the previous section, the B̈atchNormalizationm̈odel scored a 89% for its test accuracy

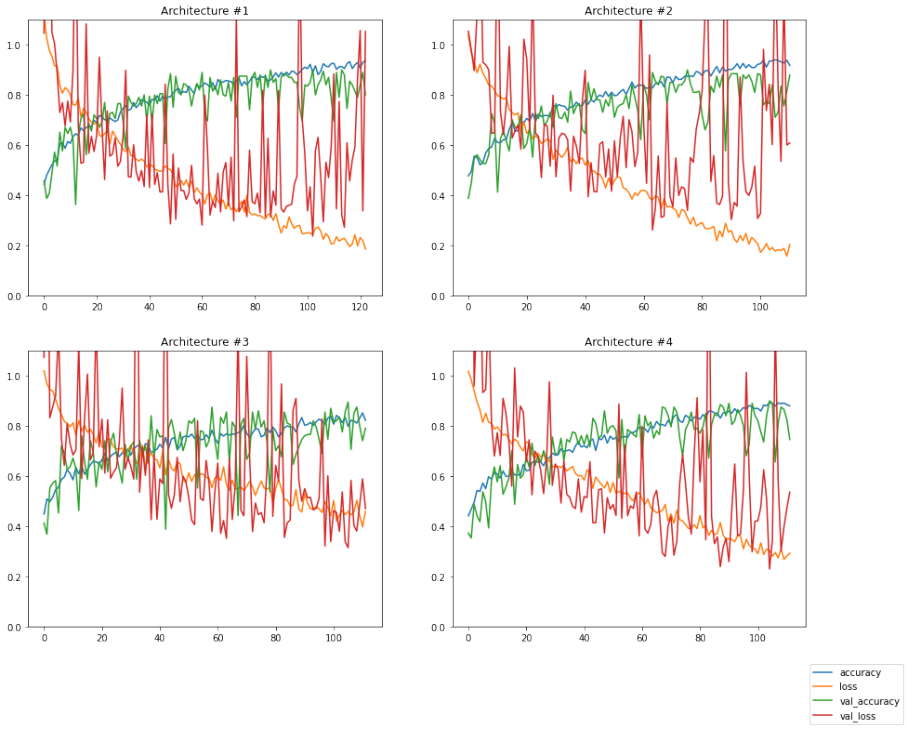


Figure 10: Learning curves of models with various residual network architectures

| Model | Average Time | Test Acc. | Test Loss |
|---|---|---|---|
| Arch. #1 | 15s | 0.82 | 0.55 |
| Arch. #2 | 17s | 0.85 | 0.62 |
| Arch. #3 | 17s | 0.80 | 0.45 |
| Arch. #4 | 18s | 0.79 | 0.56 |

Figure 11: Overview of residual network architectures used and their corresponding accuracies.

and 0.31 for its loss. Architecture #1 does not perform as well as this, but all of the tried architectures had higher test accuracies than the other approaches in the previous section. This shows how residual networks can provide more accurate models in some cases.

It should also be noted that the overall training time for residual neural networks is much lower than the previous section. Their average time per epoch was still about the same, but it took much less epochs for all of the residual networks to level off. This is another upside to using residual neural networks.

# 8 Experimenting with Transfer Learning using VGG16

The usage of pretrained models has many benefits in deep learning. One such benefit is that the amount of time it takes to train a new model with an existing model is drastically smaller than starting from scratch. On top of this, individuals who do not have access to large amounts of computing power and other such resources can use powerful existing models to create powerful models of their own. This can be done by using transfer learning.

Transfer learning is when all or a portion of a pretrained model is then appended to a new, untrained classifier. Essentially, the pretrained model will process the incoming data, and the newly appended layers will complete the task of processing and classification. This is obviously very powerful, as it allows individuals

to use very powerful models like VGG16 to buff up their existing models. In this section, experiments will be conducted to determine how to maximize validation and test accuracy while using transfer learning with the VGG16 pretrained model.

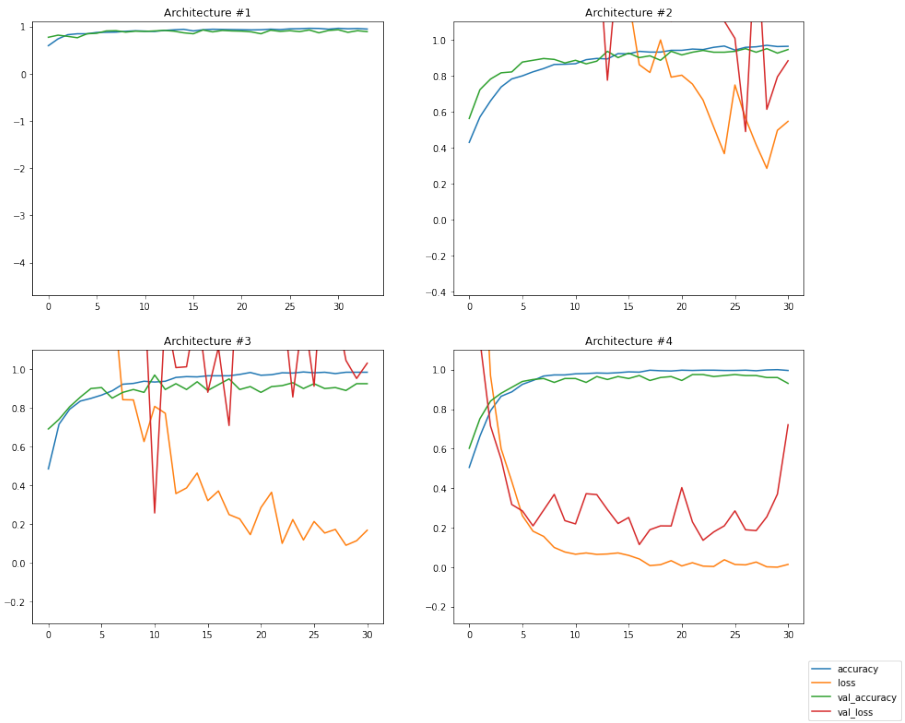


Figure 12: Learning curves of models with various transfer learning architectures

## 8.1 Architecture #1

This architecture has all of the VGG16's layers locked and about 8,000,000 trainable parameters. This model achieves a very high validation and training accuracy, at nearly 100% as shown by Figure 12. Though the accuracies are high, it's loss is horrible, with it not even appearing on the graph. As shown in Figure 13, the test accuracy of this model was 90%, while the loss was 25.21. This high loss may be due to all of the VGG16's layers being locked, so moving forward various amounts of layers will be trainable. It also took about 18 seconds per epoch to train, which is understandable considering the number of trainable parameters.

## 8.2 Architecture #2

This architecture has the last two layers of the VGG16 model unlocked for training. While it's training and validation accuracies don't achieve as high of a score as the previous model, its validation and training loss are much lower. The test accuracy was the same as the previous with 90%, but the loss was much lower at 1.26. This loss is still relatively high, so the goal is to continue to lower the loss. This model took approximately 17 seconds per epoch to train. This is surprising since the number of trainable parameters was increased from the last architecture.

## 8.3 Architecture #3

Architecture #3 has the same architecture as the previous one, but with a dropout rate of only 0.1 instead of 0.5. This change decreases the test accuracy to 87% and raises the test loss to 1.82. During training,

this model does begin to overfit between epoch 5 and 10. The training loss also drops drastically, while the validation loss remains quite high. This tells us this model does not achieve some kind of generality.

| Model | Average Time | Trainable Params | Test Acc. | Test Loss |
|---|---|---|---|---|
| Arch. #1 | 18s | 8,389,635 | 0.90 | 25.21 |
| Arch. #2 | 17s | 10,749,443 | 0.90 | 1.26 |
| Arch. #3 | 18s | 10,749,443 | 0.87 | 1.82 |
| Arch. #4 | 19s | 15,469,059 | 0.96 | 0.19 |

Figure 13: Overview of transfer learning architectures used and their corresponding accuracies.

### 8.4 Architecture #4

In this architecture, the last five layers were unlocked for training. This extra bit of flexibility allowed the model to do very well with its test accuracy reaching 96% and the loss dropping to only 0.19. It can also be seen that the issue of overfitting that occurred in the last architecture was remedied, as the validation loss more closely follows the training loss. This shows that this model is more generalized than the last three models. This model also took the longest to train, with an average of 19 seconds per epoch.

### 8.5 Choosing the Best Transfer Learning Architecture

It is quite obvious Architecture #4 performs the best out of the four. Not only does it have the highest test accuracy of 96% and lowest test loss of 0.19, but it also has the highest generality. It can be shown that this model achieves some sort of generalization, because the validation accuracy and the validation loss fairly closely follow the training accuracy and loss. And when compared to the other architectures, none of their validation losses closely follow the training loss.

It should also be noted that just because we saw a high amount of time for each epoch for this architecture, that the overall time to train this model was much less. It took approximately 25 epochs to achieve a very high validation and test accuracy. Although it took 19 seconds per epoch, this equates to a much smaller amount of time when compared to the Residual Neural Networks in the previous section. These networks took nearly 100 epochs to achieve it's highest validation and test accuracy. This shows how powerful transfer learning can be.

## 9 Conclusion

After all the experiments, two models have stuck out the most as being the best performing. The "Batch-Normalization" model from Section 6 and Architecture #4 from Section 8 not only perform very well, but also are very impactful in different ways. The "BatchNormalization" model may not have performed quite as well as Architecture #4, but this model was built completely from scratch and with the use of just a single GPU. It exemplifies how capable, proper deep learning techniques can solve this problem reasonably well, even without massive amounts of hardware and data to help train it.

Architecture #4 on the other hand shows how powerful transfer learning can be, as well as how promising this use case is. Since the pre-trained model VGG16 was trained on such a diverse and large dataset, it provides a network that can be beneficial to many scenarios. Just by unlocking a few of its last layers and creating a new classifier, a much more capable model was created, and the total training time was drastically reduced.

In the end, it is safe to say that the creation of a model which can identify individuals based on which military organization they belong to is possible. In order to scale up this project and make a more robust system which could identify more than just three organizations, one would need a larger and more diverse dataset, but this report shows it is indeed possible.