# Classification of basic troops from each of the main factions from the video game Mount&Blade II: Bannerlord, using convolutional neural network

Sebastijan Dominis

Faculty of informatics in Pula, University of Pula

16.6.2024.

## Abstract

Mount&Blade Bannerlord is one of the most popular medieval-themed video games ever created. It features six main factions, namely Sturgia, Vlandia, Battania, the Empire, Aserai, and Khuzait. These factions have many different kinds of troops, but the basic ones belonging to each of them are Sturgian Recruit, Vlandian Recruit, Battanian Volunteer, Imperial Recruit, Aserai Recruit, and Khuzait Nomad. This project aims to create a classification model that would be able to differentiate between these six troops. The data was gathered manually for this purpose, and the implementation of convolutional neural network was used. The model ultimately reached around 92.50% accuracy in its predictions. The code that was written to create it, as well as the dataset, can be accessed here: https://github.com/Sebastijan-Dominis/bannerlord_classification_model .

# 1. Introduction

Mount&Blade II: Bannerlord (from here: “Bannerlord”), created by TaleWorlds, is one of the most popular medieval-themed video games to ever exist. The main factions that appear in this video game are the Empire, Khuzait, Vlandia, Battania, Sturgia, and Aserai, and the basic troops belonging to these factions are called Imperial Recruit, Khuzait Nomad, Vlandian Recruit, Battanian Volunteer, Sturgian Recruit, and Aserai Recruit. These six types of troops have been used as the classes for this model. Each of these six factions, and their corresponding classes alike, resembles some historical culture from the Middle Ages. Vlandia resembles the Holy Roman Empire, Battania resembles the Celtic part of Europe, Sturgians resemble the Vikings and the Russians, the Empire resembles the Byzantine Empire, Khuzait resembles the Mongolian Empire, and Aserai resembles the Arab world. The facial appearance of the troops belonging to these factions is in accordance with this as well. Furthermore, the color and the style of clothing is a major distinction. Vlandia is recognized by the color red, which may appear in different brightness, the Empire is known for its various variations of purple, the Aserai is known for the yellow color, Battania for green, Sturgia for blue, and Khuzait for light blue. The in-game map is relatively large, and features various terrains and settlements, all of which have a distinct visual appeal that also depends on the weather, the time of the year, and the time of the day. Most of the troops used as classes in this model generally use the same weapons, such as the sickle, or a small sword. Battania has a specific weapon that the Battanian Volunteer may sometimes spawn with – a double-handed maul. The factors that make this classification model a bit difficult to create will be discussed in the “Dataset” section, where the methods used for gathering the data, that is the images, will also be explained in detail. Their dimensions will also be mentioned, as well as the number of images gathered in total. The next chapter will give a brief overview of the existing models, and will be followed by the “Methodology” chapter, where the aforementioned “Dataset” section appears first. That section will be followed by a section called “the Model”, and another one, called “Solutions”. A detailed review of the training process will be described in the next chapter called “Training”, where each of the sections will represent one iteration of the training process. The paper concludes with the “Conclusion” chapter, which summarizes it.

## 2. The Existing Models

Given the fact that this model is focused on a very specific topic, there are no other major models that are directly comparable to it. However, a model that was used for transfer learning in one of the iterations will be described in this chapter, as it is directly connected to the training process. That model is MobileNet. MobileNet is a class of convolutional neural networks (CNNs) optimized for mobile and embedded vision applications. It achieves this optimization by employing depthwise separable convolutions, a factorization technique that significantly reduces the number of parameters and computations required. MobileNet is particularly well-suited for applications where computational resources are limited, such as mobile applications, embedded systems, and edge computing.

Models that were used on datasets containing clothes, or military-related themes, will be discussed as well, but the datasets themselves will be discussed before that, to explain how they relate to the dataset that is the topic of this paper. Fashion-MNIST is a popular dataset that contains 60000 training images, and 10000 testing images, each representing one of the ten clothing categories, such as T-shirts, trousers, and dresses. Given the fact that the classes of our dataset differ primarily in terms of their clothing style, this dataset shows some similarity to it. Another major clothes-related dataset is DeepFashion. It includes over 800000 images with rich annotations, such as category labels, bounding box information, and landmarks. For military applications, SAR (Synthetic Aperture Radar) imagery is crucial, due to its ability to capture detailed images, regardless of weather conditions. Given that our classes are soldiers in a video game, these two show some similarity. Our dataset also shows some similarity to the combat scenarios captured, and other actions in military training captured on video.

LeNet-5 is one of the earliest convolutional neural networks, developed by Yann LeCun and his colleagues in 1998. Designed primarily for handwritten digit recognition (MNIST dataset), LeNet-5 consists of two convolutional layers, followed by two subsampling layers, two fully connected layers, and an output layer. It was not only used on the MNIST dataset itself, but also on Fashion-MNIST, thus being relevant to be mentioned here as well.

ResNet, short for Residual Network, is a deep neural network architecture introduced by Kaiming He et al. in 2015. It has proven itself to be effective on Fashion-MNIST and DeepFashion datasets. The key innovation of ResNet is the introduction of residual blocks, which allow the network to learn residual functions with reference to the layer inputs, effectively addressing the vanishing gradient problem encountered in very deep networks. ResNet models come in various depths, including ResNet-50, ResNet-101, and ResNet-152, where the number indicates the total layers.

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman in 2014. It is known for its simplicity and uniform architecture, consisting of 16 weight layers (13 convolutional layers and 3 fully connected layers). It also consisted of multiple max-pooling layers. The network uses very small (3x3) convolution filters, which are applied multiple times to capture complex patterns. VGG16 has also proven its effectiveness on Fashion-MNIST, and DeepFashion datasets. Many of the models tried during the training process discussed in this paper have primarily been inspired by the VGG16 architecture, given how simple, yet effective it truly is.

Some of the models used on datasets that regard military application include YOLO, Faster R-CNN, 3D CNNs, and LSTM. Given the nature of the military's needs, these models are primarily focused on detection, instead of only classification. They are mentioned, due to their importance in working with military-related datasets, but will not be discussed in more depth, since the nature of their goals is different, namely it goes beyond classification.

# 3. Methodology

This chapter describes the process of data gathering in detail, and gives a brief introduction to the model and the way in which the problems experienced in the creation of the final model have been dealt with. The creation of the model itself will be discussed in more detail in the following chapter.

## 3.1. Dataset

One of the features of the Single Player aspect of Bannerlord is the option to play custom battles. These battles can be randomly generated, and are executed in a way where two sides fight each other, that is the player and his army fight against the opposing army, which is controlled by the computer. One aspect of randomness is the number of soldier, which can range from one to one thousand for each side. Another one is the type of the battle itself, and it can be siege, village, or a normal battle. There are four types of troops – infantry, archers, cavalry, and horse archers, and the ratio and specific types of troops are also randomly generated. Furthermore, the factions themselves, the weather, the time of the day, the map – that is the environment – and the time of the year all add to the randomness. In the process of gathering the images for this model, many such battles have been executed.

One of the features available within these battles is the ability to pause the game at a specific moment and take screenshots of any part of the map. This has allowed much more variability in the images, and has improved their quality as well. One of the limitations that has decreased the randomness of these battles, and by extent the data gathering itself, was the fact that different classes of this model were not allowed to appear in the battle at the same time. For example, the possibility of a Vlandian Recruit appearing in the background of the image supposed to represent the Aserai Recruit has been removed. Another important thing to mention is that some of the soldiers have been photographed (screenshot) multiple times, from different angles, but in the same position. This likely explains why the model seems to be "jumping" in its validation accuracy during training. If the model recognizes one soldier from one angle, it will likely recognize the same soldier from a different angle as well. Furthermore, one thing that may make this dataset easier is the fact that the classes differ in terms of the color of clothes, although the color is sometimes hard to distinguish, due to various lighting, and other factors that will be discussed now as well.

Most of the images show only one troop, which is the class of the model, although some of the images show other troops in the background, and some of those troops may belong to a different faction. There are different factors that have made the images hard to classify. For starters, the troops appear in very different lighting, including sunny daylight, dark night, and night, but close to fire, which acts as a source of light, and makes only one part of the soldier appear brighter. Another factor is the fact that some of the soldiers were either wounded, or in the process of getting wounded when the images were taken. For example, a soldier may appear with arrows that have penetrated him, as well as with blood stains, or blood flowing directly from a wound.

Soldiers also appear in different positions, and with various weapons, since they do not always spawn with the same ones. To make it more complicated, soldiers in siege may use siege weapons, which do not appear at all in other types of battles. The angles from which the images were taken vary greatly as well, and in many of the images, only a part of the soldier's body is visible, while his face may or may not be. For example, in one image a soldier is in the process of running away with a few arrows that have already hit him, the image shows a part of his back, and he is in the dark. In another image, an image shows a face and a part of the frontal body of a fully healthy soldier using a siege weapon in the middle of the day. All of this variability contributes greatly to the complexity of the model.

In total, 6000 images were gathered – 1000 from each class. The images were gathered in a way that ensures that the scenes and the soldiers appearing in the training set, validation set, and testing set are entirely different. This means that the random battles were generated, and the images were collected firstly for the training set. After that, entirely new battles have been generated, in order to gather the images for the validation set. Finally, the same was done for the testing set of images, therefore ensuring that the model is not learning the scenes, or the specific soldiers, but the classes themselves. The ratio in which the images were split is 70:10:20. This means that each class contains 700 images in the training set, 100 in the validation set, and 200 in the testing set. Given the fact that the images were collected separately, they were not split during the coding process, but were merely imported, and given appropriate labels.

The images were scaled from HD resolution to 256x256 pixels. Smaller resolution has been considered, but the quality of the images portraying soldiers in darkness decreased so greatly when using 128x128 resolution that some of these images would even be hard for most humans to classify in such a low resolution. Therefore, this idea ultimately had to be abandoned. One of the images belonging to each class from the dataset can be seen here:





## 3.2. Model

The models tested during the training process hereby have largely been inspired by the VGG16 architecture, as mentioned earlier. Many different hyperparameters have been tested before reaching a conclusion that the final model is close to the best one that can be achieved with the CNN architecture. CNN models have the ability to spot details that even the human eye may find difficult to notice, and have therefore performed well in the given task, reaching a relatively high accuracy by the end of the training process.

## 3.3. Solutions

As mentioned in the previous two sections, the problems of this dataset regarding the classification of images are not trivial, but the CNN architecure tends to perform well in solving them. The six classes all have different style of clothing, different ethnicity-based facial features, and sometimes even different weapons, as previously described. However, the weapons are usually the same, faces are often not visible on the images, parts of the body that include clothes may be ommitted, and the darkness makes everything harder to see. Still, the convolutional neural network can find the details, even in this difficult setting. The resolution that has ultimately been decided on is one that does not waste an insane amount of resources during training, but is also not so small that the images are difficult for the human eye to differentiate. Most of the models, including the one that was ultimately chosen as the best one, are moderately deep, including a dozen layers of convolution, and a few MaxPooling layers. The models will be discussed in more detail in the following chapter.

# 4. Training

This chapter discusses the training process that was implemented as a means of creating a good classification model. The goal was to reach at least 80% accuracy, given the many problems that this dataset comes with, but also the easing factors that the CNN model experiences, all of which have been mentioned earlier. The images were put in the 256x256 resolution before importing, so there was no need to adapt their resolution within the code itself. They were imported from Google Drive, and all of them were rescaled in the ImageDataGenerator, using the following line of code: ImageDataGenerator(rescale=1./255). Various degrees of data augmentation have been used via ImageDataGenerator, and will be discussed in more detail in the following sections. A function to display the images was created, in order to ensure that the ImageDataGenerator is working as intended, and that everything is set up well, and it was all indeed set up well.

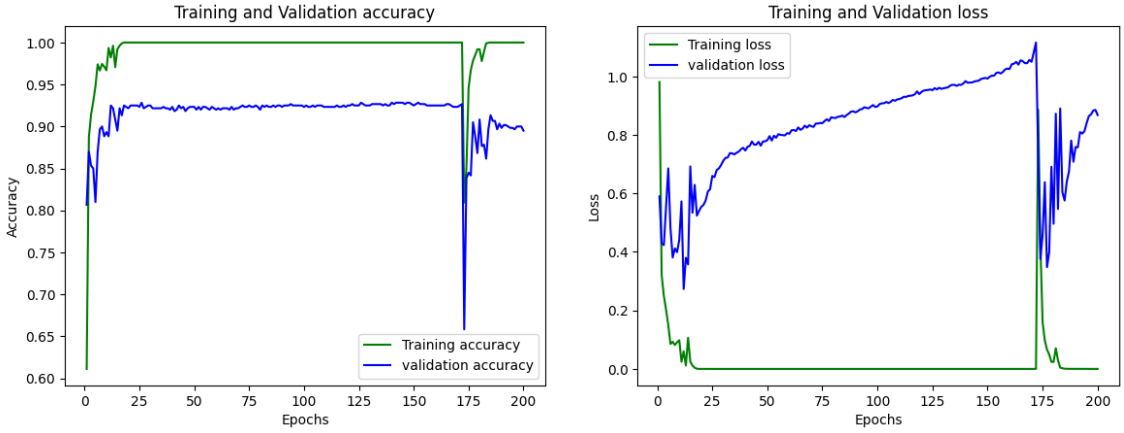
Furthermore, "adam" optimizer was used in every iteration, as it is considered to be the best. Softmax activation function was used by the end of every model, as it is viewed as a standard choice. The "shuffle" parameter of the ImageDataGenerator was always set to "True", in order to produce the most reliable results possible. Batch size was either 32, or 128, and will also be discussed in more detail in the following sections. Most of the models were run for 200 epochs, but some were only run for 100, and all of the latter models were created by utilizing the method of transfer learning.

## 4.1. Model 1

As mentioned earlier, the basis for all of the models was largely inspired by the VGG16 architecture. Therefore, the size of the filters in the convolutional layers was set to 3x3. This first model contained four of these layers, whereby the number of filters gradually increased, as it did with all of the other models that will be discussed as well. Hereby, the first convolutional layer consisted of 32 filters, the second one had 64, and the third and the fourth one had 128 filters each. Four MaxPooling layers were put in the model as well, in order to reduce the number of parameters, without making a model so deep that nothing can be learned, especially considering that the dataset contains only 6000 images, which is a far cry from the tens of thousands of images that some of the more well-established datasets with similar number of classes have. The relu activation function was chosen initially, as it is considered to be a good choice when creating a deep learning model. The learning rate was not specifically set for this model, as the only goal was to confirm once again that everything is set up well.

The results were decent, with the model reaching nearly 93% accuracy on the validation set. This can likely be explained by the fact that the classes differ in terms of the color of their clothing, which the model may be able to learn easily. That being said, this model was overfitting. The difference between its training and validation accuracy in the best epoch is nearly 6 percentage points, and it is even larger in the other epochs, with the exception of the first few, where they largely overlap, but have much lower values. The model peaked in its performance around the $20^{th}$ epoch, which is most likely due to its simple architecture. Another noticeable thing is that

both the validation accuracy and loss start to “jump” aggressively after epoch 172. Some of the “jumping” can be observed in the first two dozen epochs as well. All of this is probably due to the nature of the dataset itself. The validation images contain some instances whereby the same soldier was screenshot multiple, as mentioned earlier. If the model recognizes this particular soldier in once instance, it will likely recognize him in the other ones as well. The graphs showing this model’s performance can be seen here:

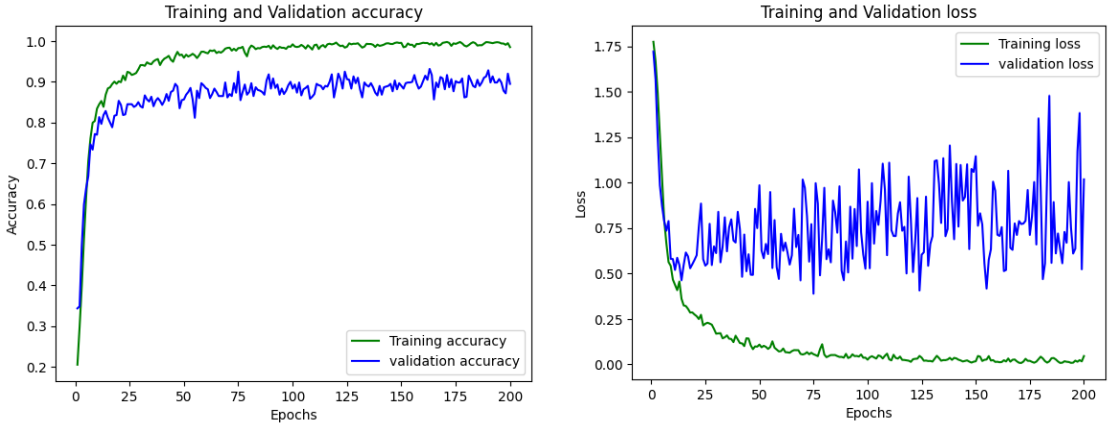



## 4.2. Model 2

The previous model confirmed that everything works, but that particular architecture is not the one that was to be ultimately used. Therefore, Model 2 is deeper, and was inspired by VGG16, as was mentioned multiple times earlier. The number of convolutional layers used hereby was eight. While the VGG16 architecture uses 13 convolutional layers, the dataset we are working with is not very large, and therefore a deeper architecture may not be appropriate to use. Still, it was expected that a slightly deeper architecture would work at least a little bit better. The first convolutional layer comes with 32 filters, the next two have 64, the two that come after them have 128, and the final three convolutional layers have 256 filters. The gradual increases are thought to be a good choice, and were used in the previous model as well, whereby they still managed to produce somewhat decent results, despite that model being extremely simplistic. Model 2 also has 4 MaxPooling layers, and uses the relu activation function. The learning rate was set to 0.00008. It was thought that some of the “jumping” in the previous model could be explained by an inappropriate learning rate, even though the nature of the validation set itself was the main culprit. Data augmentation was also used for Model 2, and the choice for the specific parameters was fairly standard. Rotation range was set to 40, width shift range, height shift range, shear range, and zoom range were set to 0.2, and the horizontal flip was included. All of this was implemented with a goal in mind – to avoid the overfitting that was evidently present in Model 1. Batch size was set to 128, as another parameter that could potentially reduce the “jumping” of validation accuracy and loss.

Surprisingly, this model did not perform any better than the previous one. Overfitting is evident, and the “jumping” was only somewhat fixed in validation accuracy, but not in loss, which

continued to “jump around” wildly throughout the epochs. It can be concluded that this phenomen is really mostly due to the nature of this dataset, and that these various hyperparameters cannot fix it. Still, the choice of a slightly lower learning rate seemed to have been a good idea. The training accuracy increased a bit more gradually, resulting in the overfit not appearing as early. Overall, this model did not perform as expected, and the results can be seen here:

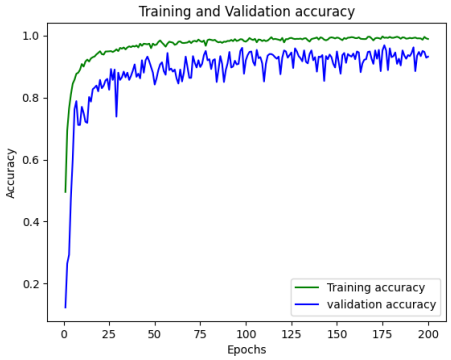


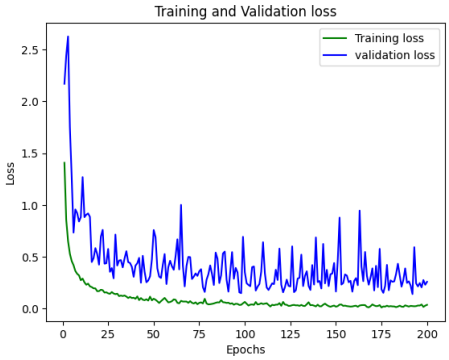

## 4.3. Model 3

The previous model did not really fix much. Model 3 aimed to fix where Model 2 failed. Data augmentation was slightly increased. The value of rotation range was increased from 40 to 45, while the values of the other parameters were increased from 0.2 to 0.25. Additionally, batch normalization layers were added throughout the whole model. Finally, a moderately strong dropout layer of 0.25 was added at the end of the model. An attempt was also made to improve the overal performance, which is why the activation function was changed from relu to leaky relu. Batch size was brought back to 32 to once again confirm that the “jumping” of the validation graphs portraying loss and accuracy throughout the epochs is simply due to the nature of the dataset itself.

The results can be described as good. Validation accuracy jumped above 96% in a few epochs, and the loss was fairly low as well. The “jumping” did not increase compared to the previous model, likely confirming our hypothesis that this phenomenon occures due to the nature of the images in the dataset. It is possible that batch normalization played a role as well, but it is not very likely, since the “jumping” on the graphs seems to be very similar to the previous model, despite the batch size being reduced from 128 to 32. In fact, there is less “jumping” on the graph that shows validation loss throughout the epochs. This may have something to do with batch normalization, or even the leaky relu activation function making the model perform better overall. A mild overfit is still present, but it is much more acceptable. Another good thing is that this model grows gradually, and only starts to get somewhat close to its peak after the first 75 epochs or so. It continues to improve mildly, reaching its best performance around epoch 175. It

seemed like this model could be improved even further with more methods of dealing with overfitting. The results of Model 3 can be seen here:

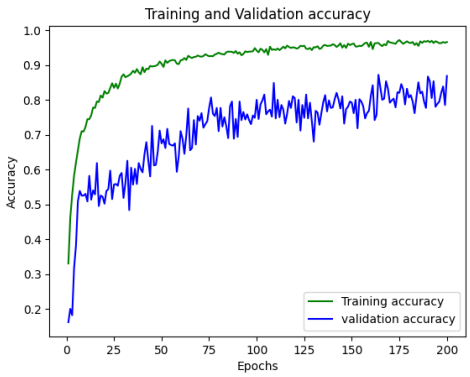


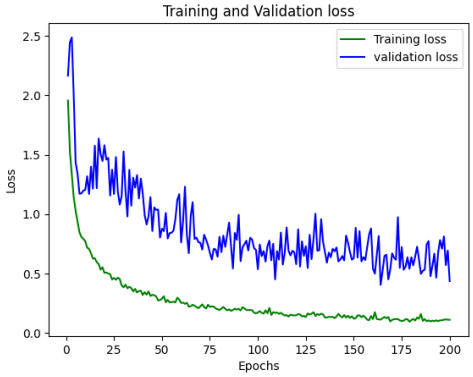




## 4.4. Model 4

The previous model seemed to perform fairly well, but still shows some mild signs of overfitting. Model 4 aimed to fix that by increasing data augmentation, and adding more dropout layers. To be more specific, rotation range was increased from 45 to 90, the other parameters from 0.25 to 0.4, and in addition to that, vertical flip was added. Model 4 also has three dropout layers with the value of 0.25, while the last dropout layer's value was increased from 0.25 to 0.5. The expectation was that the overfitting would vanish entirely. Model 3 barely had any overfitting, and Model 4 has significantly stronger data augmentation and dropout layers.

That being said, the results can possibly be described as terrible. The overfitting increased significantly, while the overall performance of the model dropped. This behaviour of the model is odd, but it could be explained by the fact that this model is relatively complex relative to the size of the dataset we are working with, which contains only 6000 images. Model 4 has eight layers of convolution, batch normalization after each layers, four layers of dropout, four MaxPooling layers, and a relatively aggressive data augmentation. It is possible that this is simply an "overkill" for the dataset at hand. It is also possible that Model 3 accidentally "jumped" above 96% accuracy in a few of its epochs, simply due to the dataset itself, which was described many times before in this paper. Either way, the performance of Model 4 is dissapointing, and can be seen here:

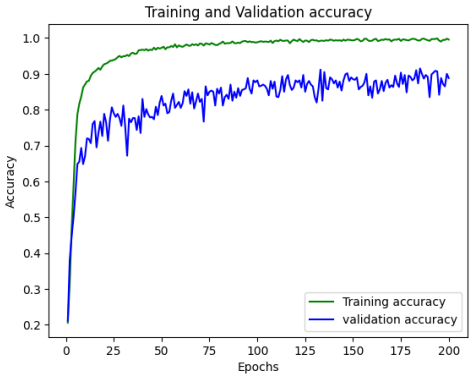


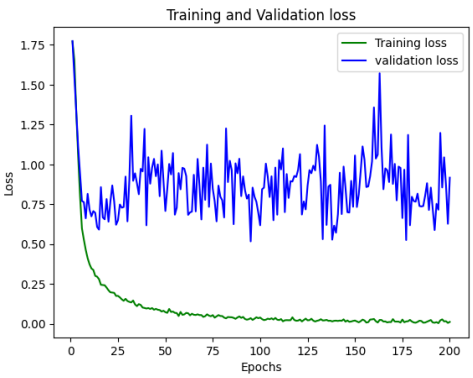



## 4.5. Model 5

The previous model performed surprisingly bad, and that could be due to its complexity. Therefore, Model 5 aims to simplify the model as much as possible. Model 5 has only six layers of convolution. The dimensions of the filters are still 3x3. The first layer of convolution has 16 filters, the second one has 32, the third has 64, the fourth and the fifth layers have 128 each, and the last layer of convolution has 256 filters. The gradual increase is still viewed as a good approach. Batch normalization was removed. The model has six layers of MaxPooling, which result in it having only around a million parameters, compared to the previous few models that had over 17 million. Model 5 also has six dropout layers, each one with the value of 0.2. Leaky relu activation function was used again, and it was used in every model from here onwards, so it will not be mentioned anymore. The alpha parameter of leaky relu was also set to 0.1, as it was in every model before and after, and this will also not be discussed anymore. Data augmentation was used, but mildly. Rotation range was set to 20, width shift range, height shift range, shear range, and zoom range were set to only 0.1, and only horizontal flip was left as well. The learning rate was increased to 0.0001, as the learning of the previous model was growing gradually, but was almost too slow at the learning rate of 0.00008.

Model 5 did not improve anything. In fact, it was the worst model thus far. The validation accuracy barely even touched the 90% mark, while the training accuracy was getting very close to 100%. The “jumping” on the loss graph was the most noticeable thus far. It seems like the simplification of the model did not do anything to improve it. Reducing data augmentation, and removing batch normalization only worsened the model, despite it being more simplistic in terms of the number of the layers of convolution, and the number of parameters. It seems like some aspects of overfitting may indeed be due to the nature of dataset as well. That being said, Model 3 had much fewer issues with overfitting. Therefore, this simple, fifth model had to be refused. The results of Model 5 can be seen here:

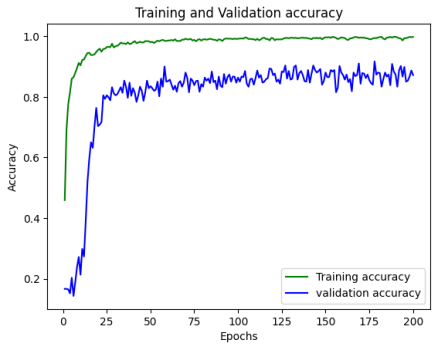


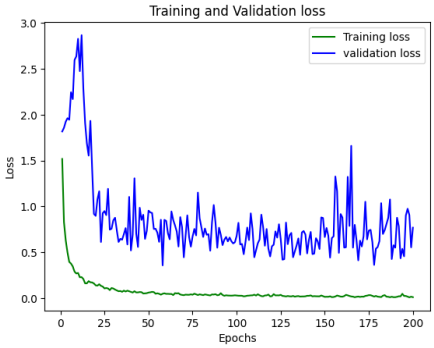




## 4.6. Model 6

Given that the previous model performed so badly, while Model 3 performed fairly well, a variation of Model 3 was implemented again. However, one last check was made to confirm that nothing can be done regarding the "jumping" on the validation graphs. To do that, everything was left the same as it was in Model 3, except the batch size. The architecture of the model, the data augmentation, the learning rate etc. are all the same in Model 6. Only the batch size was increased from 32 to 128, to see what kind of effect it would have.

The performance of the model actually reduced. The model reach just over 91% validation accuracy in one epoch, which is roughly five percentage points lower than the best epoch from Model 3. This also implies that the overfitting increased, since the training accuracy still got very close to 100%. Meanwhile, the "jumping" of the model was not removed, nor decreased. This shows that increasing the batch size in this model does not do anything good for it, and that the dataset, both in terms of size and content, has a strong effect on the results we are getting. This is the first model that was also tested on the testing set. Best epoch in terms of its performance on the validation set was chosen, but this model only reached around 84.75% accuracy and 0.8824 loss, which is not terrible, but is still a far cry from the 90% that definitely seemed possible. The only thing we learned from this iteration is that increasing the batch size has a negative impact on our models, due to the dataset we are working with. The results of Model 6 can be seen here:

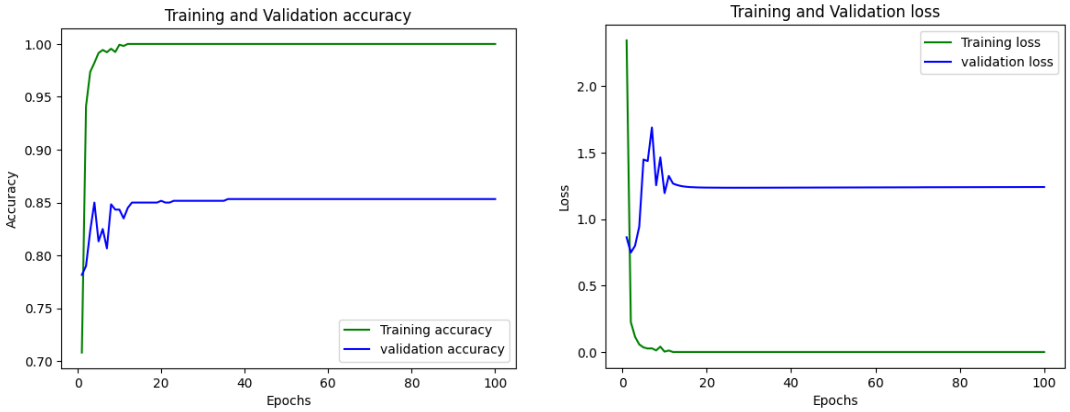



## 4.7. Model 7

Before settling with the results we got from Model 3, a decision was made to see how the model performs when the method of transfer learning is implemented. For this purpose, a popular model called MobileNet was used. MobileNet was briefly described earlier in this paper. For Model 7, the learning rate was set to 0.0001, and this is the first model that ran for only 100 epochs, as it contained no data augmentation or dropout layers, and it was expected that the model would reach its peak performance a bit earlier than epoch 100.

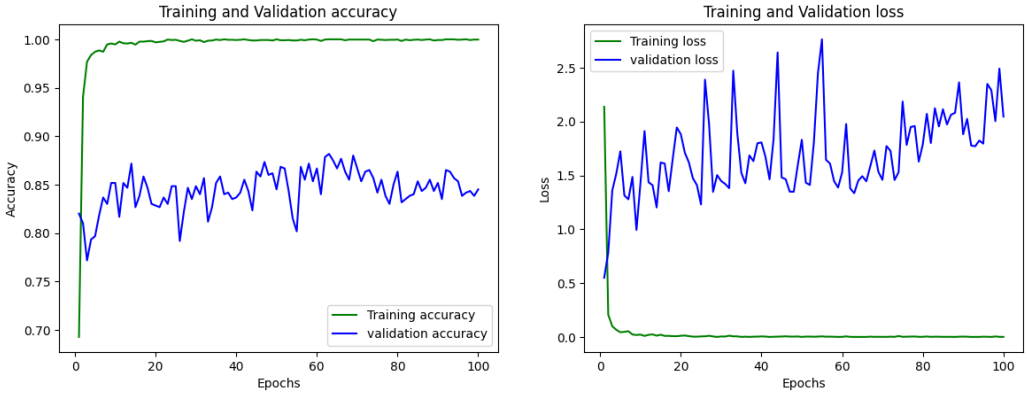
Indeed, the model basically peaked slightly before epoch 20, and only improved marginally in the following two dozen epochs. The validation accuracy seems to stagnate around 85%, while the training accuracy reaches a full 100%. The difference between the training and validation loss is also visible. All of this implies an aggressive overfit. The results of this first model that utilized transfer learning can be seen here:




## 4.8. Model 8

The previous model performed fantastically well on the training set, but showed clear signs of overfitting. Model 8 takes Model 7 and adds a relatively strong dropout layer of 0.5 after the flatten layer, and before the densing layers. The rest is left intact, in order to see how the model's behaviour changes when a relatively powerful dropout is added.

The validation accuracy of this model definitely got above 85% in many of its 100 epochs, but never even reached 90%. Not only that, but the "jumping" of the model suddenly became wild again, and the loss gradually increasing throughout the epochs. Therefore, we cannot really say that this model performs much better than the previous one. The results of Model 8 can be observed here:

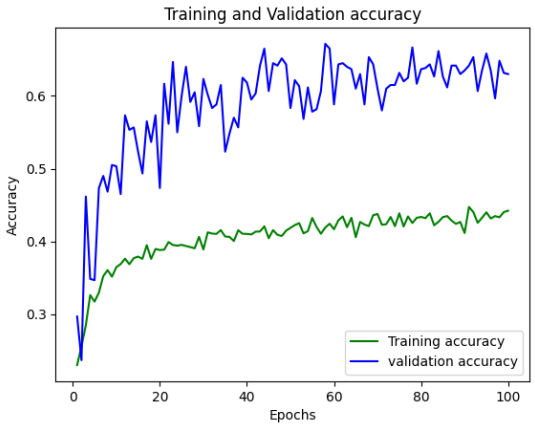


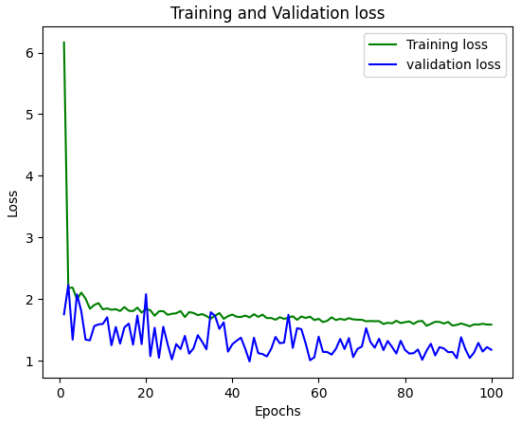

## 4.9. Model 9

The previous model clearly did not do much to remove the problem of overfitting. Model 9 aims to do so with a very aggressive implementation of data augmentation. Rotation range was set to 90, while width shift range, height shift range, shear range, and zoom range were all set to 0.5. Both horizontal flip and vertical flip were implemented. In addition to that, brightness range

ranging from 0.2 to 1.0 was added, as well as channel shift range with the value of 150. The dropout layer of 0.5 was left as well, and the batch size was set to 128, simply to see how the model behaves with that batch size, given the small likelihood that the transfer learning may make it behave a bit differently.

The results are nearly shocking. The underfitting of this model can easily be described as wild. It reaches over 66% accuracy on the validation set, while barely even getting close to 45% on the training set, and that is only if we take the best performing epochs. The underfitting can also be observed in the loss graphs, whereby the validation loss is definitely lower than the training loss. This means that it is possible to deal with the issue of overfitting on this dataset, albeit very challenging. That being said, we did not yet know what the true price of completely fixing the overfit on this model is, given that Model 9 simply gave us the opposite issue – underfitting. The results can simply be explained with the fact that the data augmentation used hereby was very aggressive, and the results themselves can be observed here:



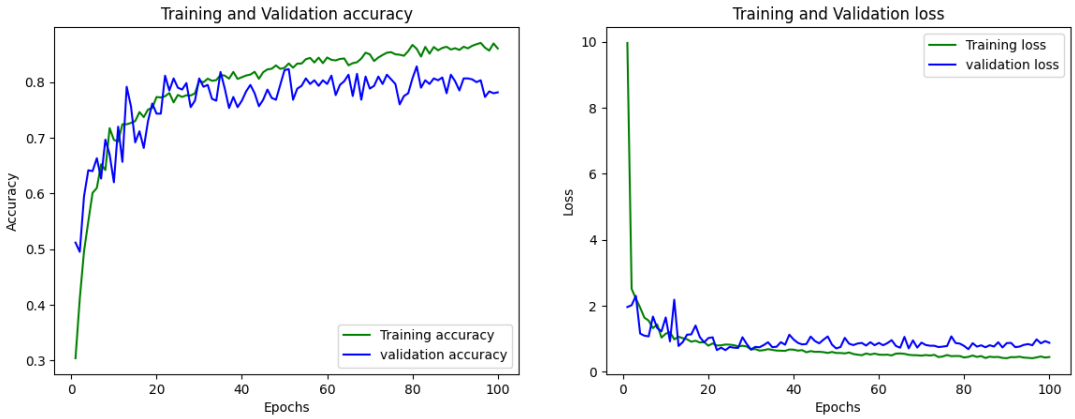




## 4.10. Model 10

The previous model showed obvious signs of underfitting, whereas the two models before it showed clear signs of overfitting. This model aims to find the right balance, in order to fix both of these issues, and create a smooth transfer learning model that performs as well as it can. For this purpose, the value of the dropout layer was reduced down to 0.3, while the overall data augmentation was made more moderate too. Rotation range was set to 60, width shift range, height shift range, shear range, and zoom range were set to 0.3, and only the horizontal flip and vertical flip were left as well. The rest was left the same.

Indeed, the problems experienced in the previous models were removed, but at a considerable cost. The validation accuracy barely got close 83% in one of the epochs, and mostly hovered around 80%. Therefore, this iteration shows that it is possible to virtually remove the issue of overfitting on this dataset, but only at the expence of the overall model's performance dropping considerably. Given that our ultimate goal is good performance on the testing set, this seems to be suboptimal, meaning that the tradeoff is too large, and it may be a better choice to accept

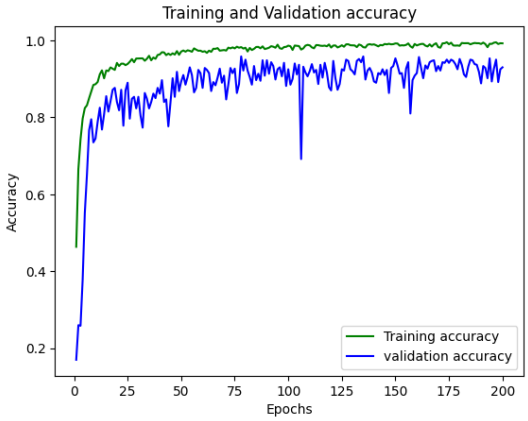
some degree of overfitting, if it gives us better results on the testing set. We have also learned that transfer learning does not yield us better results on this dataset. The results of Model 10 can be observed here:

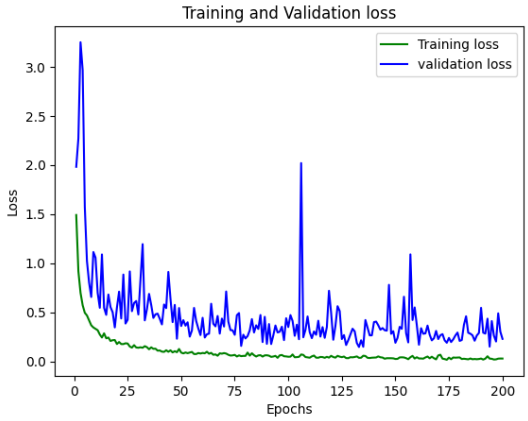



## 4.11. Model 11

Given everything we have learned about the behaviour of this dataset thus far, Model 11 takes Model 3, and increases the data augmentation parameters only a little bit. Rotation range is increased from 45 to 60, while width shift range, height shift range, shear range, and zoom range are all increased from 0.25 to 0.28. Horizontal flip is left as well. Batch size is still set to 32, since Model 6 taught us that increasing it only reduces the results that we ultimately get from the model. The architecture itself is also left the exact same as in Model 3, as we mentioned.

Model 11 performed relatively similar to Model 3, as we would expect, although its performance seemed a bit lower, somewhat surprisingly. It only reached 95.67% validation accuracy in its best epoch, which is around one percentage point less than the best performing epoch of Model 3. Still, this result is deemed acceptable, which is why the model of that epoch was saved and tested on the testing set. There, the model reached a fairly satisfying 92.5% accuracy, and 0.2653 loss. This means that if we were to put this model in production, it would perform fairly well. That being said, Model 3 did reach one percentage point higher, which is why we had to test that model on the testing data as well, before deciding which one is better, and whether any additional improvements can be made. The results of Model 11 can be seen here:




## 4.12. Evaluating Model 3, and deciding on the best model

Given that Model 11 never reached the highest accuracy score of Model 3, it was decided that Model 3 would be retrained, whereby the model would be saved at each epoch, and the best epoch would be chosen to be tested on the testing set. This was done, and Model 3 indeed reached 96.5% validation accuracy in one of the latter epochs.

That being said, the model from that particular epoch scored only 90% on the testing set. This is a full 2.5 percentage points lower than the results we got from the best epoch of Model 11. It is possible that Model 11 is slightly more complex, due to greater data augmentation, and thus less adapted to the validation set itself, while performing better overall. Model 3 accidentally performs slightly better in a few of the epochs, but is less complex, and less prepared for the images it has not seen. Given that the experience with some of the previous models showed us that moderately increasing data augmentation and dropout only increases the overfitting of Model 3, it was decided that this would not be done. A simpler model was also tried before, and it did not perform any better. We have also found that an aggressive increase in data augmentation drastically reduces the overall performance of the model. Lastly, we have learned that increasing batch size worsens the model's performance considerably, both on the validation set, and the testing set. We know this now, because Model 6 only reached 84.75% accuracy on the testing set, while Model 3 reached 90%, and Model 11 reached 92.5%. With all of that in mind, it seems like nothing can be made to improve this model any further, at least not within the confines of the CNN architecture. Therefore, the best-performing epoch of Model 11, namely epoch 161 was chosen as the best model for this particular dataset.

# 5. Conclusion

A decent classification model for classifying the basic troops belonging to each of the main six factions of Bannerlord has been created within the scope of this project, and it reached around 92.5% accuracy on the testing set.

That being said, the dataset has some constraints. Some of the soldiers have been screenshot multiple times, resulting in the “jumping” of the graphs portraying the performance of the validation set. The dataset also contains only 1000 images per class, which is decent, but still far less than the 10000 often seen in the more established datasets.

Furthermore, only six of the troops from Bannerlord have been classified here, since classifying all of them would require a massive amount of data and resources, both of which was beyond the scope of this project. Thus, a future model could try to classify all of the troops, or perhaps take the existing dataset from here, add more data, and attempt to create a segmentation model. This model could show where the soldier is, where his weapon is, as well as the buildings, horses, trees, etc.

All of that being said, this model performed fairly well for its purpose.

# Literature


1. Howard, A.G., et al., 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint* arXiv:1704.04861.
2. TaleWorlds Entertainment, 2020. *Mount&Blade II: Bannerlord*. [video game] (Microsoft Windows).
3. He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
4. LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp.2278-2324.
5. Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
6. Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).
7. Ren, S., He, K., Girshick, R. and Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).
8. Tran, D., Bourdev, L., Fergus, R., Torresani, L. and Paluri, M., 2015. Learning spatiotemporal features with 3d convolutional networks. In Proceedings of the IEEE international conference on computer vision (pp. 4489-4497).
9. Xiao, H., Rasul, K. and Vollgraf, R., 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.
10. Liu, Z., Luo, P., Qiu, S., Wang, X. and Tang, X., 2016. DeepFashion: Powering robust clothes recognition and retrieval with rich annotations. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1096-1104).
11. S. Mahdianpari, B. Salehi, W.L. Chang, A. Mohammadimanesh, and Y. Zhang, 2018. The First Wetland Inventory Map of Newfoundland at a Spatial Resolution of 10 m Using Sentinel-1 and Sentinel-2 Data on the Google Earth Engine Cloud Platform. Remote Sensing, 10(9), 1354. doi:10.3390/rs10091354.