Image Classification of Playing Card Suits Using CNN & Transformer

By Michael Sullivan, Elliot Marsico, Travis Stauffer, & Brady McDermott

CMPSC 448

Penn State University

December 2, 2023

Task, Dataset, & Preprocessing

Task:

For this project, our task was to build an image classification system that could identify the suit (clubs, diamonds, hearts, or spades) of playing cards when presented with an image of one. Of the three deep learning systems we had to choose from, we chose to use a convolutional neural network (CNN) and a transformer-based system to build this project. We chose to not use a recurrent neural network (RNN) because it isn't often used for image classification and wouldn't work as well as the other two options. We decided to use the Python library named Tensorflow to help us with the machine learning aspect of the project since it has many built in functions related to CNN's and transformers. To be more specific, we decided to try and use features within this library and others such as Keras, which provides a Python interface for artificial neural networks, MobileNet, which provides models for training classifiers, ViT, which measures relationships between input images, and more.

Dataset:

The dataset that we chose to use was found on a website called Kaggle, which is a website for the data science community. One of the features of this website is that people create and upload large datasets that are meant to be used for machine learning projects. Here we found a dataset containing 7,624 images of all types of playing cards. The original purpose of this dataset was to use it to identify both the suit and the number/type of the card, however we decided to classify just the suit of each card in the set. Because of this, we eliminated the joker card from the dataset since it doesn't have a suit and had lower quality images anyways. All images in this dataset are 224 x 224 x 3 and in JPG format. They are also cropped so that they only show a single card and the card takes up at least half of the pixels of the image. This was important to us because we needed to have high quality images to train our model on that didn't have too much extra noise in them. Because of the size of this dataset, a link to the dataset is included in the ReadMe file on GitHub.

Preprocessing for CNN:

In designing a CNN for image classification, a lot went into preparing the images for the model itself. With the help of tqdm, we were able to load all of the card images and their corresponding filenames into python lists. Initially all of the card images were randomly loaded into a pandas dataframe, with the joker set being dropped due to low data quality. To capture absolute randomness, we proceeded to shuffle the data set to avoid any human interference. The data was split into test and train data, the latter of which would eventually be split into validation data. The overall data split was 60, 20, 20, which we felt was appropriate for the complexity of this task. We decided to include a test set separate from the validation set, because we ended up implementing a callback system that used the validation accuracy as a parameter. We designed a function that created an image generator object based on the train and test dataframe. The Tensorflow ImageDataGenerator made it extremely easy for us to augment images and even allowed us to split up the data into training and validation sets. Due to the random behavior of these generators, we had to make sure the test generator was not shuffled so we could compute performance metrics. Using an rgb color scheme, we decided to include color in the data to assist in identifying suits. Because we observed varying data augmentation techniques worked well with each model, we decided to create multiple instances of the data generators. We felt that the data generators were perfect for this task, because they allowed us to capture additional randomness in the test set.

Preprocessing for Transformer:

In our transformer model development, we meticulously preprocessed our dataset, which started with the random scrambling of a large collection of images. The large data set helped to mitigate the risk of class weight imbalances. This was followed by an 80-10-10 split into training, validation, and test sets. Each image was processed using OpenCV (cv2), where they were read in three color channels—red, green, and blue—and resized to 200x200 pixels for uniformity. Each image was then normalized, as it scales pixel values to a standard range, aiding in the efficient processing of data by the transformer. We then employed the 'patchify' library to split each image into an 8x8 grid of patches. These patches were subsequently reshaped into flat patch arrays with dimensions 8x8x3, aligning them with the transformer model's input requirements. Finally, we labeled these patches according to their respective classes. This comprehensive preprocessing strategy, which includes techniques like image resizing, normalization, patchification, and labeling, ensures the input data is optimally prepared, laying a robust foundation for effective model training, evaluation, and ultimately, high performance and accuracy.

Implementation & Architecture of the Two Systems

CNN:

This model was built using a relatively basic CNN architecture. The CNN was composed of three convolutional layers. This design choice was based on our use of 224x224 pixel images, allowing for efficient downsizing of the image after each convolutional layer without incurring excessive decrease in size. Each convolutional layer incorporated a ReLU activation function to introduce non-linearity, followed by max pooling to emphasize the most critical features from the feature maps. We proceeded to the convolution phase by flattening our input into 1D arrays. We added one more 64 width dense layer to capture complex relationships between the features. To finish things off, we dropped out half of the data set to mitigate the risk of overfitting, followed by a softmax to represent the output as probabilities. The final layer had 5 neurons with probabilities corresponding to each of the classes. The model implemented an adam optimizer with categorical cross entropy loss. The Adam Optimizer is a common choice for CNNs due to its ability to handle higher dimension parameter spaces. In fitting the model, we created a set of callbacks that we felt would prevent the model from incorrectly updating weights in the training process. For the size of the images and data set, we felt no more than 10 epochs were appropriate due to time constraints and lack computational resources. Despite the satisfactory performance of our model, we recognized the limitations posed by our dataset's size. We decided to implement transfer learning with MobileNetV2, hoping to see an increase in accuracy. MobileNetV2 is a tensorflow CNN model that was trained on millions of images. Along with the MobileNet model, we added an additional 64 neuron fully connected layer, and an output layer. Additionally, we implemented this model with average pooling on the last convolution. The weights of the model were initially set to those that had been set when the model was pretrained on the imagenet dataset. The model was fitted using the same callback method mentioned above.

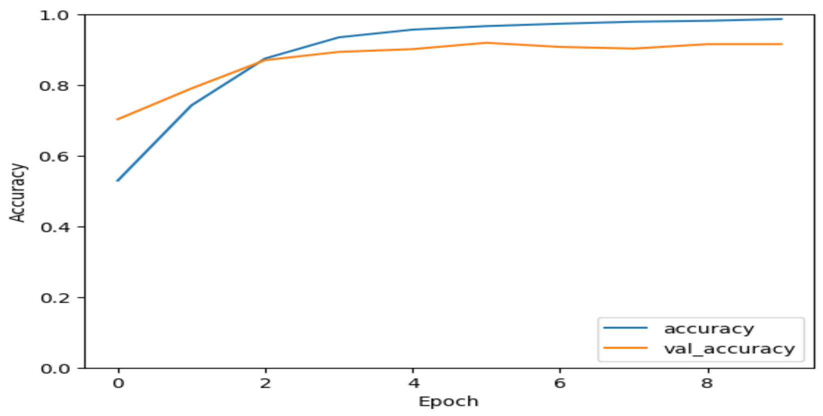
Transformer:

In our Vision Transformer (ViT) implementation, we approached image classification by converting images into a sequence of flattened 2D patches, diverging from conventional CNN methodologies. The key component of the Vision Transformer (ViT) is its array of transformer encoder layers. Each of these layers includes a MultiHeadAttention layer, which enables the model to focus on different parts of the image at the same time, and a feed-forward network for processing the information further. A distinctive element of our ViT is the inclusion of a 'Class Token', a trainable embedding concatenated with patch embeddings, which aggregates information across transformer layers for final classification. Layer normalization and dropout are strategically applied within the MLP and attention layers to improve training efficiency and prevent overfitting. Each encoder layer, following a normalization-attention-feed-forward pattern with residual connections, ensures robust and deep layer stacking. The output from these layers, after a final layer normalization, leverages the Class Token representation for classification through a softmax layer. This integrative approach, combining MultiHeadAttention, MLPs, and Class Tokens with advanced concepts like layer normalization and residual connections, equips the ViT to capture both localized features and global image representations, establishing it as a potent tool for image classification tasks.

Training The Models

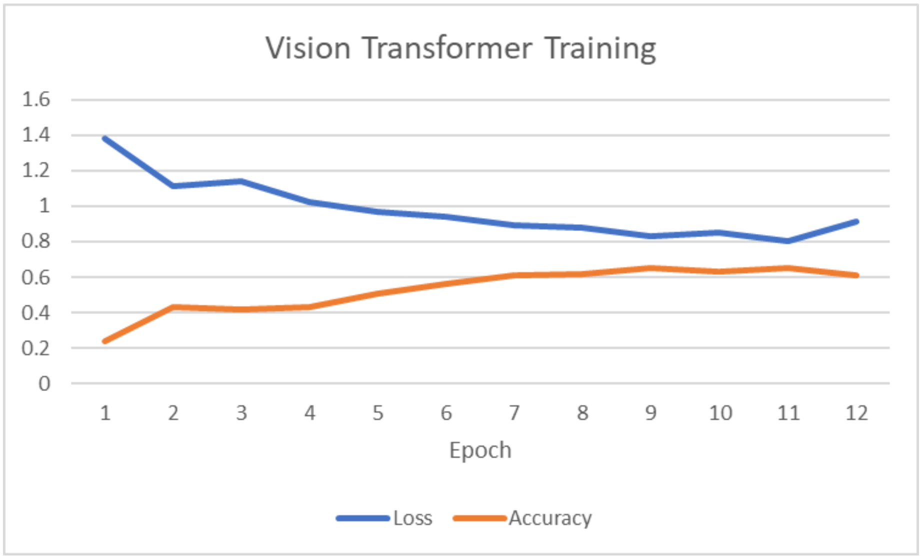
CNN:

The model that we designed took very long to fully train on the dataset. With a batch size of 32 and 10 epochs, the model took over an hour to train. Surprisingly, the accuracy on the training set was slightly below the validation accuracy for a significant amount of time during the training process. An interesting observation during training was the strange relationship between an increased validation loss and an increasing/stagnant validation accuracy. We believe this to be a good indication that the model made some very confident incorrect predictions during training, which drove the validation loss up.This led us to believe that convergence had already occurred, and additional training would have negligible impact on results. While we did include a callback to monitor the behavior of the model, there was no halting and 10 full epochs were executed. For the pretrained model, we decided that 12 epochs was more appropriate to get the most accurate model possible. During the training process, the validation accuracy showed very little change, and as expected, the accuracy and loss showed incremental improvement. The training stage took about 35 minutes, and it concluded without stopping early.



Transformer:

In the training phase of our Vision Transformer (ViT) model, we faced significant challenges primarily due to the model's complex nature and our constrained computational resources, which prolonged the training process considerably, with each epoch lasting between 45 minutes to an hour. This extended training time inherently limited our ability to fully fine-tune and optimize the model, consequently affecting the final accuracy. After completing 12 epochs, the model achieved a test accuracy of approximately 65 percent. We adopted categorical cross entropy as our loss function, a standard metric in multi-class classification models, to evaluate the model's performance across epochs by comparing the predicted outputs with the actual labels. At the conclusion of these 12 epochs, the model reported a validation loss of around 0.8, indicating its generalization capability on unseen data. To optimize the learning process, we utilized the Adam optimizer, which is renowned for efficiently managing sparse gradients and dynamically adjusting learning rates for individual weights. This optimizer is particularly effective for complex models like ViT, as it allows different sections of the network to converge effectively at varying learning rates, thereby shaping the overall training dynamics and outcomes of our model.



Results, Observations, & Conclusions

Results & Observations in CNN:

Our model showed very important signs of good accuracy and resistance to noise. The training finished with a validation accuracy of 0.9159 and a validation loss of 0.3376. Its 91.59% accuracy on the test set was a great sign of performance on two completely separate, and shuffled data sets. An F1 score of 92% indicated to us that there was a strong balance between precision and recall. Additionally, the model had very balanced performance between each of the classes despite having slightly different amounts of data points. As mentioned above, there was a strange relationship between accuracy and loss on the validation set. We believe this to be due to spades being very confidently classified as clubs.

For our pretrained model, we observed some interesting results. The Transfer Learning approach led us to a validation accuracy of 0.8372 , and a validation loss of 0.7385. Once again, MobileNetV2 performed similarly on both the validation and test set, indicating a strong ability to effectively classify a wide range of data. We observed through trial and error that shifting, shearing, and flipping helped the model to better fit to the task at hand. After attempting various image sizes, we noticed that 224*224 image size was needed to capture the true suit of a card. We went back and forth on whether or not to disclude color from our dataset, however we observed that color was a useful tool in identifying class patterns. We decided to include a dropout layer before the softmax output, and we noticed that a 0.5 dropout rate decreased overfitting significantly in our model. After completing our personal model, we observed that keeping our images at high resolution, while including three convolutional layers allowed us to recognize key features without overfitting. Through our efforts in designing an effective CNN model, we eventually recognize the key component of machine learning to be quality of data. The chess dataset had too few images for any convolutional neural network to truly notable patterns.

Results & Observations in Transformer:

In our exploration of the Vision Transformer (ViT) model, we encountered various challenges and successes that provided valuable insights. Notably, using the original dataset with approximately 400 images proved inadequate for the ViT model to converge successfully. Attempts to enhance model performance by increasing the number of patches and the resolution of images (up to a maximum of 224x224 pixels) were unsuccessful in addressing the issues faced. Furthermore, various adjustments such as adding more layers, increasing batch sizes, and modifying the dropout rate and learning rate, did not yield significant improvements in accuracy. The data's imbalance posed another challenge; despite our efforts to incorporate class weights to offset this, the improvement in model performance was still negligible.

On a positive note, when we transitioned to a larger dataset, specifically for playing cards with around 7000 images, we observed a marked difference. The same model, applied to this enlarged dataset, showed enhanced performance, although this came with the trade-off of a substantial increase in training time, approximately 4-5 times longer than with the smaller dataset. This observation underscores the critical impact of dataset size on the efficacy of complex models like ViT, balancing the trade-offs between model accuracy and computational demands.

After completing 12 epochs, the model achieved a test accuracy of approximately 65 percent.

Overall Conclusions:

After training and running the two models on the dataset, we found that the CNN model is better at classifying the playing card suits than the transformer model. As explained above, the overall accuracy of the CNN model, both the regular one and the pretrained one, were better than the accuracy of the transformer model. The results look like this for multiple reasons. One main reason we think the CNN was so much better is because of its efficiency. CNN's are much faster to run and computationally inexpensive when compared to transformers, which are very expensive to run, especially with higher quality images. If we had the computational power to run more epochs with the transformer model, we think it would have performed even better than it did. CNN's are also better at capturing spatial hierarchies from the images via convolutional layers. A transformer system needs to use self-attention mechanisms to relate the image patches together, which can require a lot of training data. We also saw that our pretrained model didn't perform as well as our own model for CNN. We think this is because we were able to fine-tune our own model more towards the data than the pretrained one could do. Overall, both models performed very well.

Challenges & Obstacles

The main challenge that we faced during this project was with our dataset. Our dataset had to be changed multiple times throughout the process of building the model. Initially, we considered something simple, such as a binary classification problem to distinguish between images of dogs and cats. We thought this was too easy and then found a dataset with images of five different chess pieces. We thought that this was a good plan at first, but once we built our two models, our results weren't that good. We were able to conclude that these issues were mostly because of the dataset being too small and including many images that weren't of the best quality. Our solution to this problem was to find a much better data set containing thousands of high quality images. This led us to the playing cards idea, which is what we stuck with. This dataset contained over seven thousand high quality images (as explained above). When we trained our models with these images, our accuracy immediately went up, especially for the transformer. With the original dataset, our transformer system was barely better than a random guess. After introducing it to the new dataset, it gained a lot of accuracy.

Another challenge that we faced was with the overall performance of our CNN model. When we trained our original CNN system on the playing card dataset, its accuracy wasn't far over 67%. Although we aren't graded on accuracy in this project, we wanted to build something that was actually decent and wanted better accuracy. Our solution to this problem was asking the professor if we are allowed to use pretrained models, to which he said we could. We did some research and discovered MobileNet, which contains lightweight models that can be used to help train image classification systems. Surprisingly, MobileNet did not perform as well as expected on the new data. We believe this to be due to potential data mismatch between what MobileNet was trained on, and what data was actually fed to the model. After realizing this, we did a lot of fine tuning on our CNN model, which was described above. As we improved our parameters, the model continued to improve for both the pretrained model and our own. This showed us that our main challenge to overcome was fine tuning how our model worked, which we were able to overcome and get over 90% accuracy on our own model and over 80% accuracy on the pretrained model.

Conclusion

In concluding this project analysis, we have learned a lot about deep learning and the ins and outs of how it works. We learned a lot about the importance of high quality data, the importance of fine tuning all of the parameters within your model, and the importance of choosing the right type of deep learning system for your project. We ended with a very good success rate for our playing card dataset on both systems and are very proud of the success we got out of our models.