

Advanced Machine Learning

Final Project:

Pole vault Technique Recognition Model

05/08/2022

Lukas van der Watt

Table of Contents

Abstract.....	3
Background.....	3
Project Goal	3
Data Description and Sourcing	4
Approach.....	6
Implementation	7
Simple CNN:.....	7
Simple CNN with Augmentation & Dropout	8
Stage1.....	9
Stage2.....	10
Stage 3.....	11
Stage 4.....	12
Stage 5.....	13
Utilizing the VGG-16 CNN with ImageNet	14
Results Summary	16
Conclusion	17
References.....	17

Abstract

This project concentrates on the domain application of a deep learning method known as convolution neural network. In this project the pole vault is dissected into five stages that can accurately describe the correct implementation of technique to be successful in the event, at any level of competition. It uses an image recognition (convolution neural network) model that classifies the level of execution using a sequence of images in a vault. In this document the pole vault at Kent State University was used to demonstrate the workings and potential of analytic models that can enhance the training experience for both coaches and athletes.

Background

Sports is a major industry today and with time it has become much more than just an enjoyable past time. Sports are extremely competitive and sport team managers and coaches are turning to analytics to get valuable information about the athletes to enhance the performance of their team. This not only makes the team more successful, but it drives the potential for financial growth of a professional or collegiate program. The pole vault is an extremely technical sport and can also be very dangerous if not executed correctly. Coaches spend years honing in their eye to be able to spot subtle differences in an athlete's technique to be able to jump higher. Similar to sports like golf, the use of video recordings and images is an extremely helpful mechanism to understand the error being made as well as what to adjust in order to fix the technical mistakes. Therefore, developing a model that is trained on the pole vault technique can have a great impact on the pole vault society and benefit athletes trying to better their technical form.

Project Goal

Develop an image recognition model that can make classifications between well and poorly executed pole vault technique.

Data Description and Sourcing

This is a new domain implementation as there are no other models focused on classifying the technique execution for pole vault exclusively. With a model focused on only the pole vault technique, a suitable dataset needs to be used for training. Since there were no open-source pole vault data set available, one needed to be made from scratch.

To be able to fulfill the goal of the project a dataset focused on the technical implementation of the vault needed to be sourced. Furthermore, a successful technical jump needed to be defined in order to source the correct data that would be able to support the goal of the project. Lastly the quality and view angle were important while data sourcing to bring diversity to the table while still being able to make distinctions in certain phases of the vault. A view of the athletes back whilst they approach the mats for example, would therefore add minimum value to the data as it is difficult to make technical observations from this viewpoint.

The data was sourced from the 2020 Tokyo Olympic Men's Pole Vault Final as well as social media pages such as VaulterMagazine and Jumper's World. Self-recorded videos of collegiate competitions and practices of the KSU pole vault team was also used to train and test the model. It is therefore clear that the data was sourced from real world events. The "Polevault" dataset is a record of over 100 jumps that spans from elite to beginner athletes. It consists of about 600 images spread through 5 stages.

As mentioned before the pole vault jumps were recorded and divided into 5 stages that can provide a wholesome representation of the pole vault technique. Stage_1 involved the takeoff position where the arms are straight and one leg is extended backward to make what's known as the C-position at the small of the back. During Stage_2 the vaulter was judged on the straight line going from the end of the plant box, the position of the extended leg and the position of the top hand. Two straight arms were also required to have a successful implementation during this phase. In Stage_3 the athlete needs to have the hips in a high position and the legs either tucked against the chest or extended toward the shoulders and not the bar. In Stage_4 the athlete is in a full extended position and in-line with the pole. A rotated position was also acceptable if the previous condition was met. Stage_5 involved the position of clearance over the bar. This position had to be an actual clearance of the bar and in a good position of clearance as well from a technical standpoint. This phase is open to multiple body positions as height over the bar was the true metric.

A depiction of a good vs. bad stage can be seen below:

Good

Stage_1:



Poor



Stage_2:



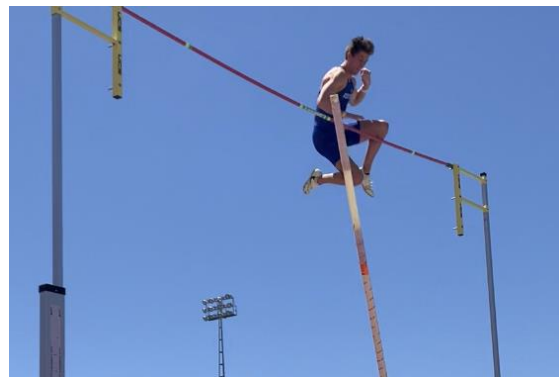
Stage_3:



Stage_4:



Stage_5:



Approach

To start, the data was split into 70% training set, 15% to validation set and 15% to the testing set. For this project an 11-layer Convolution Neural Network (CNN) was made from scratch and trained on the Polevault dataset. This is denoted as the “Simple” model throughout the project. A CNN model with more layers was also tested but was found to overfit very quickly and was later removed. A batch size of 32 and 10 was also varied in the respective datasets to see its effect on the test results. The 5 different stages were then run on the model to get the model performance on the respective stages. This made it possible to see which stage was more inconclusive or less accurate. A model with image augmentation was then run on the stages to see how this diversity in images could improve the model since the dataset was small. Lastly a final model was then trained and implemented using the pretrained VGG16 with imagenet dataset. This model proved to be more accurate, and a prediction was made on the test set. This allowed for a confusion matrix to be set up.

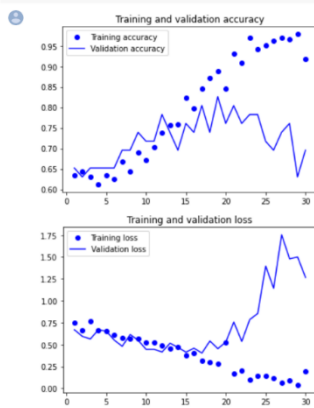
Implementation

Simple CNN:

```
[9] from tensorflow import keras
    from tensorflow.keras import layers

    inputs = keras.Input(shape=(180, 180, 3))
    x = layers.Rescaling(1./255)(inputs)
    x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
    x = layers.MaxPooling2D(pool_size=2)(x)
    x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
    x = layers.MaxPooling2D(pool_size=2)(x)
    x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
    x = layers.MaxPooling2D(pool_size=2)(x)
    x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
    x = layers.MaxPooling2D(pool_size=2)(x)
    x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
    x = layers.Flatten()(x)
    outputs = layers.Dense(1, activation="sigmoid")(x)
    model = keras.Model(inputs=inputs, outputs=outputs)
```

BATCHSIZE = 32

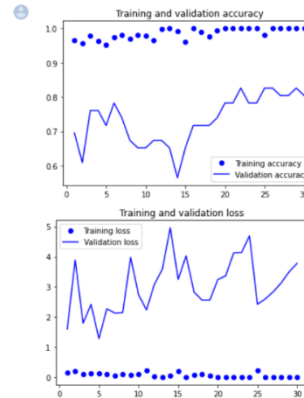


Evaluating the model on the test set

```
[17] test_model = keras.models.load_model("convnet_from_scratch.keras")
      test_loss, test_acc = test_model.evaluate(test_dataset)
      print(f"Test accuracy: {test_acc:.3f}")

2/2 [=====] - 2s 331ms/step - loss: 0.4302 - accuracy: 0.8125
Test accuracy: 0.812
```

BATCHSIZE = 10



Evaluating the model on the test set

```
test_model = keras.models.load_model("convnet_from_scratch.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")

5/5 [=====] - 2s 197ms/step - loss: 1.4346 - accuracy: 0.7083
Test accuracy: 0.708
```

In the images above we can see that the batchsize = 10 is performing great on the training set but the validation loss doesn't really improve whereas the batchsize of 32 had a clear tradeoff between the training and validation sets. An optimal epoch number can be distinguished. This batchsize number was then used in the rest of the model coding.

Simple CNN with Augmentation & Dropout

Model with entire “Polevault” dataset

Defining a new convnet that includes image augmentation and dropout

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

```
Epoch 46/50
12/12 [=====] - 38s 3s/step - loss: 0.4567 - accuracy: 0.7594 - val_loss: 0.4777 - val_accuracy: 0.7826
Epoch 47/50
12/12 [=====] - 36s 3s/step - loss: 0.4155 - accuracy: 0.7995 - val_loss: 0.6045 - val_accuracy: 0.6957
Epoch 48/50
12/12 [=====] - 38s 3s/step - loss: 0.5012 - accuracy: 0.7193 - val_loss: 0.5012 - val_accuracy: 0.7174
Epoch 49/50
12/12 [=====] - 38s 3s/step - loss: 0.4271 - accuracy: 0.7834 - val_loss: 0.4387 - val_accuracy: 0.7174
Epoch 50/50
12/12 [=====] - 38s 3s/step - loss: 0.4253 - accuracy: 0.7781 - val_loss: 0.7055 - val_accuracy: 0.7174
```

Evaluating the model on the test set

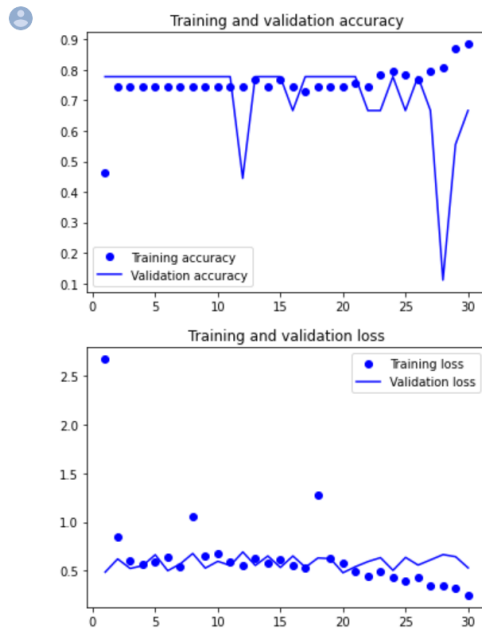
```
+ Code + Text
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")

2/2 [=====] - 2s 306ms/step - loss: 0.3919 - accuracy: 0.8125
Test accuracy: 0.812
```


Stage1

Simple (1)

```
Epoch 25/30
3/3 [=====] - 8s 2s/step - loss: 0.3875 - accuracy: 0.7821 - val_loss: 0.6362 - val_accuracy: 0.6667
Epoch 26/30
3/3 [=====] - 7s 2s/step - loss: 0.4292 - accuracy: 0.7692 - val_loss: 0.5574 - val_accuracy: 0.7778
Epoch 27/30
3/3 [=====] - 7s 2s/step - loss: 0.3401 - accuracy: 0.7949 - val_loss: 0.6111 - val_accuracy: 0.6667
Epoch 28/30
3/3 [=====] - 7s 2s/step - loss: 0.3485 - accuracy: 0.8077 - val_loss: 0.6652 - val_accuracy: 0.1111
Epoch 29/30
3/3 [=====] - 8s 2s/step - loss: 0.3149 - accuracy: 0.8718 - val_loss: 0.6432 - val_accuracy: 0.5556
Epoch 30/30
3/3 [=====] - 8s 2s/step - loss: 0.2472 - accuracy: 0.8846 - val_loss: 0.5286 - val_accuracy: 0.6667
```



Evaluating the model on the test set

```
test_model = keras.models.load_model("convnet_from_scratch.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")

1/1 [=====] - 1s 723ms/step - loss: 1.1091 - accuracy: 0.9167
Test accuracy: 0.917
```

Augmentation Model (1)

```
Epoch 46/50
3/3 [=====] - 8s 2s/step - loss: 0.5860 - accuracy: 0.7436 - val_loss: 0.5508 - val_accuracy: 0.7778
Epoch 47/50
3/3 [=====] - 8s 2s/step - loss: 0.5213 - accuracy: 0.7436 - val_loss: 0.5728 - val_accuracy: 0.7778
Epoch 48/50
3/3 [=====] - 8s 2s/step - loss: 0.4274 - accuracy: 0.7436 - val_loss: 0.6224 - val_accuracy: 0.7778
Epoch 49/50
3/3 [=====] - 8s 2s/step - loss: 0.5499 - accuracy: 0.7436 - val_loss: 0.5853 - val_accuracy: 0.7778
Epoch 50/50
3/3 [=====] - 8s 2s/step - loss: 0.4417 - accuracy: 0.7436 - val_loss: 0.5551 - val_accuracy: 0.7778
```

Evaluating the model on the test set

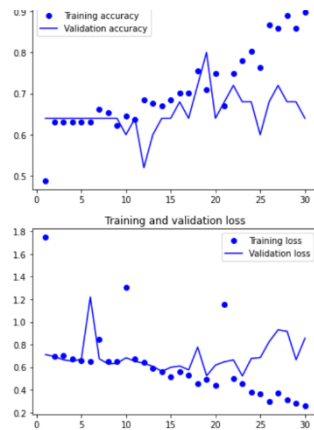
```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")

1/1 [=====] - 1s 577ms/step - loss: 0.7172 - accuracy: 0.6667
Test accuracy: 0.667
```

Stage2

Simple (2)

```
Epoch 26/30
4/4 [=====] - 12s 3s/step - loss: 0.2931 - accuracy: 0.8661 - val_loss: 0.8261 - val_accuracy: 0.6800
Epoch 27/30
4/4 [=====] - 12s 3s/step - loss: 0.3680 - accuracy: 0.8583 - val_loss: 0.9303 - val_accuracy: 0.7200
Epoch 28/30
4/4 [=====] - 12s 3s/step - loss: 0.3106 - accuracy: 0.8898 - val_loss: 0.9164 - val_accuracy: 0.6800
Epoch 29/30
4/4 [=====] - 12s 3s/step - loss: 0.2786 - accuracy: 0.8583 - val_loss: 0.6653 - val_accuracy: 0.6800
Epoch 30/30
4/4 [=====] - 12s 3s/step - loss: 0.2585 - accuracy: 0.8976 - val_loss: 0.8563 - val_accuracy: 0.6400
```



Evaluating the model on the test set

```
test_model = keras.models.load_model("convnet_from_scratch.keras1")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")

1/1 [=====] - 2s 2s/step - loss: 0.5383 - accuracy: 0.6452
Test accuracy: 0.645
```

Augmented Model (2)

```
Epoch 48/50
4/4 [=====] - 13s 3s/step - loss: 0.5356 - accuracy: 0.6772 - val_loss: 0.5795 - val_accuracy: 0.6800
Epoch 49/50
4/4 [=====] - 12s 3s/step - loss: 0.4936 - accuracy: 0.7165 - val_loss: 0.5147 - val_accuracy: 0.7600
Epoch 50/50
4/4 [=====] - 13s 3s/step - loss: 0.5478 - accuracy: 0.6614 - val_loss: 0.6217 - val_accuracy: 0.7200
```

Evaluating the model on the test set

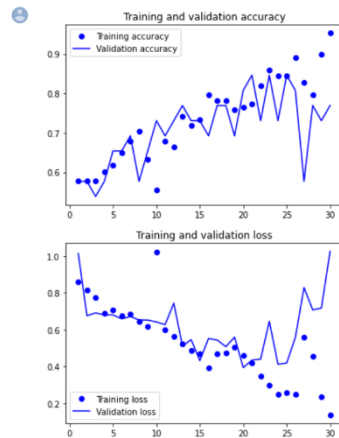
```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:3f}")

1/1 [=====] - 1s 1s/step - loss: 0.5541 - accuracy: 0.7419
Test accuracy: 0.741935
```

Stage 3

Simple (3)

```
Epoch 26/30
4/4 [=====] - 12s 3s/step - loss: 0.2474 - accuracy: 0.8906 - val_loss: 0.5580 - val_accuracy: 0.8077
Epoch 27/30
4/4 [=====] - 12s 3s/step - loss: 0.5610 - accuracy: 0.8281 - val_loss: 0.8295 - val_accuracy: 0.5769
Epoch 28/30
4/4 [=====] - 12s 3s/step - loss: 0.4546 - accuracy: 0.7969 - val_loss: 0.7079 - val_accuracy: 0.7692
Epoch 29/30
4/4 [=====] - 12s 3s/step - loss: 0.2369 - accuracy: 0.8984 - val_loss: 0.7180 - val_accuracy: 0.7308
Epoch 30/30
4/4 [=====] - 12s 3s/step - loss: 0.1370 - accuracy: 0.9531 - val_loss: 1.0259 - val_accuracy: 0.7692
```



Evaluating the model on the test set

```
test_model = keras.models.load_model("convnet_from_scratch.keras1")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:3f}")

1/1 [=====] - 1s 1s/step - loss: 0.6713 - accuracy: 0.5625
Test accuracy: 0.562
```

Augmented Model (3)

```
4/4 [=====] - 13s 3s/step - loss: 0.5313 - accuracy: 0.7031 - val_loss: 0.4893 - val_accuracy: 0.7308
Epoch 48/50
4/4 [=====] - 13s 3s/step - loss: 0.6148 - accuracy: 0.6719 - val_loss: 0.5247 - val_accuracy: 0.5769
Epoch 49/50
4/4 [=====] - 13s 3s/step - loss: 0.5009 - accuracy: 0.7656 - val_loss: 0.5225 - val_accuracy: 0.6923
Epoch 50/50
4/4 [=====] - 13s 3s/step - loss: 0.4772 - accuracy: 0.7422 - val_loss: 0.4742 - val_accuracy: 0.7308
```

Evaluating the model on the test set

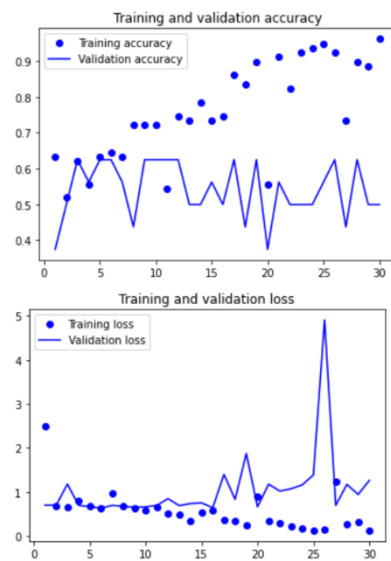
```
✓ [37] test_model = keras.models.load_model(
1s      "convnet_from_scratch_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")

1/1 [=====] - 1s 1s/step - loss: 0.5541 - accuracy: 0.7419
Test accuracy: 0.741935
```

Stage 4

Simple (4)

```
-----
3/3 [=====] - 8s 2s/step - loss: 0.1539 - accuracy: 0.9241 - val_loss: 4.9090 - val_accuracy: 0.6250
Epoch 27/30
3/3 [=====] - 8s 2s/step - loss: 1.2361 - accuracy: 0.7342 - val_loss: 0.6971 - val_accuracy: 0.4375
Epoch 28/30
3/3 [=====] - 8s 2s/step - loss: 0.2710 - accuracy: 0.8987 - val_loss: 1.1723 - val_accuracy: 0.6250
Epoch 29/30
3/3 [=====] - 8s 2s/step - loss: 0.3057 - accuracy: 0.8861 - val_loss: 0.9427 - val_accuracy: 0.5000
Epoch 30/30
3/3 [=====] - 8s 2s/step - loss: 0.1254 - accuracy: 0.9620 - val_loss: 1.2609 - val_accuracy: 0.5000
```



Evaluating the model on the test set

```
✓ [37] test_model = keras.models.load_model("convnet_from_scratch.keras1")
1s      test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")

1/1 [=====] - 1s 818ms/step - loss: 0.6232 - accuracy: 0.6316
Test accuracy: 0.632
```

Augmented Model (4)

```
Epoch 48/50
3/3 [=====] - 8s 2s/step - loss: 0.4818 - accuracy: 0.7975 - val_loss: 0.6568 - val_accuracy: 0.6250
Epoch 49/50
3/3 [=====] - 8s 2s/step - loss: 0.4503 - accuracy: 0.7722 - val_loss: 0.9276 - val_accuracy: 0.5625
Epoch 50/50
3/3 [=====] - 8s 2s/step - loss: 0.6149 - accuracy: 0.6962 - val_loss: 0.6099 - val_accuracy: 0.6250
```

Evaluating the model on the test set

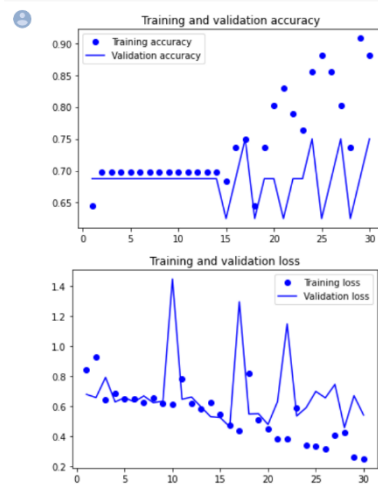
```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")

1/1 [=====] - 1s 797ms/step - loss: 0.5988 - accuracy: 0.5789
Test accuracy: 0.578947
```

Stage 5

Simple (5)

```
3/3 [=====] - 7s 2s/step - loss: 0.3116 - accuracy: 0.8553 - val_loss: 0.6540 - val_accuracy: 0.6875
Epoch 27/30
3/3 [=====] - 7s 2s/step - loss: 0.4052 - accuracy: 0.8026 - val_loss: 0.7438 - val_accuracy: 0.7500
Epoch 28/30
3/3 [=====] - ETA: 0s - loss: 0.4207 - accuracy: 0.7368INFO:tensorflow:Assets written to: convnet_from_
3/3 [=====] - 9s 3s/step - loss: 0.4207 - accuracy: 0.7368 - val_loss: 0.4558 - val_accuracy: 0.6250
Epoch 29/30
3/3 [=====] - 7s 2s/step - loss: 0.2608 - accuracy: 0.9079 - val_loss: 0.6684 - val_accuracy: 0.6875
Epoch 30/30
3/3 [=====] - 7s 2s/step - loss: 0.2480 - accuracy: 0.8816 - val_loss: 0.5396 - val_accuracy: 0.7500
```



Evaluating the model on the test set

```
test_model = keras.models.load_model("convnet_from_scratch.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")

1/1 [=====] - 1s 797ms/step - loss: 0.5296 - accuracy: 0.7368
Test accuracy: 0.737
```

Augmented Model (5)

```
Epoch 40/50
3/3 [=====] - 8s 2s/step - loss: 0.6435 - accuracy: 0.7105 - val_loss: 0.5547 - val_accuracy: 0.6875
Epoch 47/50
3/3 [=====] - 8s 2s/step - loss: 0.5612 - accuracy: 0.7105 - val_loss: 0.5937 - val_accuracy: 0.7500
Epoch 48/50
3/3 [=====] - 8s 2s/step - loss: 0.6090 - accuracy: 0.7105 - val_loss: 0.5243 - val_accuracy: 0.6875
Epoch 49/50
3/3 [=====] - 8s 2s/step - loss: 0.5251 - accuracy: 0.7368 - val_loss: 0.5017 - val_accuracy: 0.7500
Epoch 50/50
3/3 [=====] - 8s 2s/step - loss: 0.5486 - accuracy: 0.7500 - val_loss: 0.5197 - val_accuracy: 0.6875
```

Evaluating the model on the test set

```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:3f}")

1/1 [=====] - 1s 785ms/step - loss: 0.5637 - accuracy: 0.6316
Test accuracy: 0.631579
```

Utilizing the VGG-16 CNN with ImageNet

```
Epoch 7/30
13/13 [=====] - 245s 19s/step - loss: 0.0453 - accuracy: 0.9975 - val_loss: 0.5103 - val_accuracy: 0.7529
Epoch 8/30
13/13 [=====] - 245s 19s/step - loss: 0.0352 - accuracy: 0.9975 - val_loss: 0.5066 - val_accuracy: 0.7765
Epoch 9/30
13/13 [=====] - 244s 19s/step - loss: 0.0279 - accuracy: 1.0000 - val_loss: 0.5148 - val_accuracy: 0.7529
Epoch 9: early stopping
```

```
[55] model.evaluate(test_x,test_y,batch_size=32)
```

```
3/3 [=====] - 45s 14s/step - loss: 0.4527 - accuracy: 0.7978
[0.4527008533477783, 0.7977527976036072]
```

It can be seen here that an accuracy of 80% was achieved in this model. The training curves (accuracy and loss) can also be seen in the graphs below:

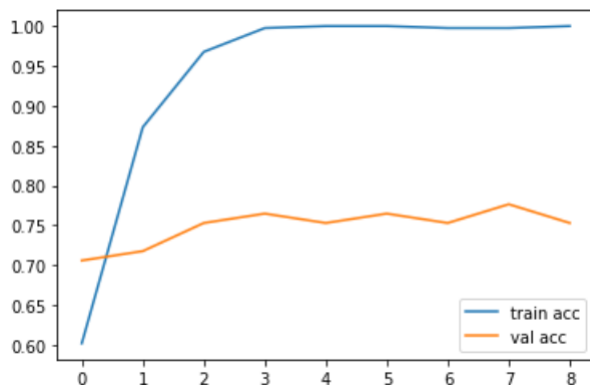


Figure1: Model Training Accuracy

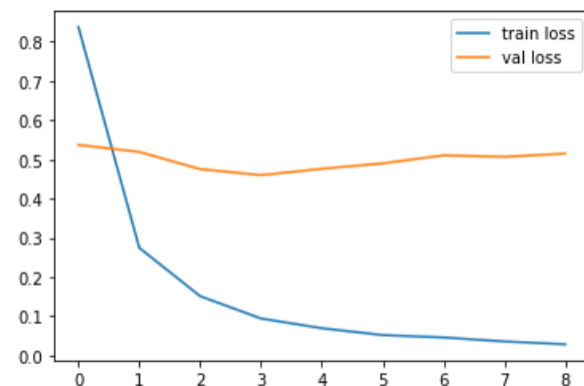


Figure 2: Model Training Loss

As the training accuracy gets higher, a slight improvement in the validation accuracy can also be seen. The validation accuracy did not get worse as the training accuracy improved and therefore the model did not overfit. Similarly in the loss curves we see a drastic drop in the training loss with the early epochs, whilst the validation loss also slightly decreases. We see a slight increase in the validation loss around 3 epochs which could indicate that the model is starting to overfit. Luckily an early stop was coded in into the model which stops the training of the model after a steady increase of the validation loss is observed, The early stop was set to wait 5 epochs after the increase is initially observed effectively preventing overfitting.


		precision	recall	f1-score	support
 {'Bad': 0, 'Good': 1}	0	0.64	0.78	0.70	27
	1	0.89	0.81	0.85	62
accuracy				0.80	89
macro avg		0.76	0.79	0.77	89
weighted avg		0.82	0.80	0.80	89

Table 1: Model Confusion Matrix

In this confusion matrix we can see that there were 27 images in the test set classified as “Bad” and 62 images as “Good”. The F1- score is the harmonic mean of the precision and recall. Precision is the fraction of correctly classified instances among the instances classified as positive. Recall (a.k.a Sensitivity) is the true positive rate meaning the proportion of positives correctly identified. We get a pretty good recall for both cases. The F1 score of the “Good” classification is higher than that of its counterpart and this could be due to the representation in the Polevault dataset. It is natural occurrence that better/ good jumps occur more often and therefore it has a larger representation in the dataset with a slightly higher f1- score. If a classification threshold would be included this slight bias could be addressed. With an accuracy of 80% we can conclude that the model is successful in its goal to distinguish between good and poor technique implementation. This is not ideal in the real world as it has an error percentage of over 10%. However, there is potential for this model to be more accurate as more data is added and a classification threshold is put in place. This model will be able to have a relevant impact in the classification of pole vault technique.

Results Summary

Table 2: Results Summary

Dataset	Accuracy		
	Training	Test	Augmentation with dropout (Test)
Stage1	88.6	91.7	66.7
Stage2	89.8	64.5	74.5
Stage3	95.3	56.2	74.5
Stage4	96.2	63.2	57.8
Stage5	88.2	73.7	63.2
Simple Model	77.8	81.2	-
Pretrained Model	99.7	79.8	-

In the table above it is clear that the Simple model had the most trouble in identifying Stage 3. This insight allows for revision of the images in this particular set and an attempt could be made to either get better quality images in this stage or get images that more accurately describes the difference between the classes. However when looking at the test results of the augmented model in this phase it drastically increased to 74.5% which indicates that it could be a lack of images (dataset is too small) or diversity of the images in this stage.

When looking at stage 4 we can also see that this stage had the highest accuracy during training and had a scanty overall performance in the simple test and augmented test sets. We could therefore see that some overfitting might have occurred during this stage. A similar analysis approach could be made for each of the stages.

Finally we can see that the Simple model performed the best with an accuracy of 81.2%. However when a prediction was made on the test set there was a bias observed in the classification as more often the “Good” classification was made. When the same prediction was run on the pretrained model it was clear that there was less bias in the model as there were representation for both classes. For this reason the Pretrained model is superior and more accurate.

With classification problems it is important to test the validity of the model around the decision boundary. To address this, predictions were made on the test set and a confusion matrix was constructed to evaluate the model performance and the position of the current decision boundary. With an accuracy of 80% and an average F1- score of 78% we have a working predictive model that can successfully distinguish the different techniques and therefore can conclude that the decision boundary is in a position that allows the model to have predictive power.

Conclusion

Through this analysis it has been shown that the domain knowledge in Pole Vault was successfully applied to a deep learning model that had predictive power. A champion model was identified and weaknesses in the model stages could clearly be seen, allowing the modeler to make adjustments in the dataset for future applications and model designs. This model has potential to be implemented in the form of an app that could then be sold to coaches at universities and high school as well as the athletes themselves. The project goal was therefore achieved but the model could benefit from improvements such as increased data size and a more in-depth analysis of classification in the model. An attention map for instance, explaining which pixels are most important in classification could be cross referenced with the actual theory of technique implementation to see how valid the model's decision-making is.

References

1. Pole Vault Technique Model : [Code](#) (available on GitHub)
2. Jumpers'World Instagram (Data Source)
3. VaulterMAGazine Instagram (Data Source)
4. 2020 Tokyo Oympics Pole Vault Final Videos: (Data Source)
<https://www.youtube.com/watch?v=Uyuw1tX0gK8>