CSI4106

Intro. Artificial Intelligence

Final Project Report

Testing Convolutional Neural Networks with Sports Image Classification

Group 22

Ryan McCarron 300078968

Colin McFarlane 300074534

Will Lennox 300071951

Apr 22, 2022

Abstract

In this report we investigate different convolutional neural network (CNN) classification models to classify sports images. We tested the strengths and weaknesses of different models when tasked with training from the same dataset. We used known CNN models DenseNet121, MobileNet, MobileNetV2, ResNet50, and a custom model we created to use as a baseline. Our results concluded that MobileNet was the strongest model with our dataset.

Introduction

Our goal with this project is to find a strong CNN model for classifying sports images. We chose this project topic as a way to learn more about what makes efficient and accurate classification models. All members of our group are new to machine learning and we wanted to learn how different model architecture affects classification results. The topic of sports images provided us with many unique classes to choose from. As part of our evaluation we wanted to determine which sports the models struggled with classifying. For instance, we expected the models to identify 'swimming' images easily as it is visually very different from other sports like basketball or volleyball. We experimented with models created by researchers such as the MobileNet model. Our group also developed our own simple classification model to contrast the results from the third-party models. We used our model as a baseline performance when comparing all models used. The image dataset was gathered from Kaggle [7]. We reduced the dataset size from 22 classes to 16 classes. Our team screened the images and removed irrelevant and incorrect images from the dataset. This left us with 10094 images in our dataset.

This report will discuss the results of the different models when trained and tested with a sports image dataset. The report will outline background information on CNN and the models selected. We will explain our custom model and performance in comparison to the other researcher's models. The results of our project will be summarized and a final discussion will highlight our findings.

Background and Related Work

For our solution we will be using a variety of different convolutional neural network (CNN) models. CNN are widely used machine learning models that excel in image recognition and classification. CNN typically consists of a mix of convolution layers, pooling layers, and dense layers. The convolution layers take an input matrix and it is sliced into smaller sub matrices. Convolution operations using a filter are applied to the sub matrices. The output of these mathematical operations results in an output matrix the same size or smaller than the input. Pooling layers are typically placed after convolution layers. They reduce the given matrix by taking the max or average value on slices of the matrix. Once again resulting in a smaller output matrix. Finally, dense layers are hidden layers that connect every neuron from one layer to all the neurons in the next layer. Using an activation function, the input neuron values are

processed and sent to the next layer's neurons. These layers end the CNN and provide the final classification output.

To evaluate our models, we kept track of the training accuracy and loss, validation accuracy and loss, and the test accuracy and loss. Loss measures how far the model's label predictions were on average from the actual labels. The accuracy measures the percentage of correct label predictions out of all the predictions made. Confusion matrices were used to show a summary of prediction results on our test dataset. The matrices contained data for the accuracy, precision, recall and F1 score. Accuracy is the percentage of correct predictions made on a dataset. Precision is the number of positive class predictions that were correct. Recall is the proportion of positive class predictions. F1 score is a weighted average between precision and recall. We also noted the time it took to fit the models for metrics.

Proposed Solution

The first model we considered was a custom model we created ourselves that we compared to widely known research models. This model would be a sequential model and would consist of a combination of convolutional two dimensional layers and two dimensional max pooling layers as well as a flatten layers and dense layers, with a softmax activation to assist with multi class classification. The architecture of our model is shown below, in Figure 1.

```
model = tf.keras.Sequential([

  layers.Conv2D(16, 3, padding='same', activation='relu',input_shape=input_shape),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(16, activation='softmax')
])

learn_rate = (0.001)
optimizer = tf.keras.optimizers.RMSprop(learning_rate=learn_rate)
```

Figure 1 - Custom Model

All the models were run with the same hyperparameters. For all of the models we tested we used the RMSprop optimizer with an initial learn rate of 0.001 in order to be consistent. We used a batch size of 16, 20 epochs and 466 steps per epoch. Validation steps were 25 with a frequency of 1.

This model would serve as a baseline that we would use to evaluate the following research models: MobileNet, DensetNet121, ResNet50 and MobileNetV2. MobileNet is a CNN that was created by researchers at Google that consists of 25 layers. The architecture for mobilenet is shown in Figure 2.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 2 - MobileNet Architecture (Howard et al., 2017, p.4)

The MobileNet CNN is more complex than the model we created with significantly more layers than the base model. However, it is more lightweight than some of the other models we will test. MobileNet utilizes a unique form of convolutional layers. It splits a typical convolutional layer into two different layers, a depthwise convolutional layer and a pointwise convolutional layer. The unique style of convolution layers does not make MobileNet less powerful, “MobileNet uses 3 × 3 depthwise separable convolutions which uses between 8 to 9 times less computation than standard convolutions at only a small reduction in accuracy” (Howard et al., 2017, p.3). We were interested in the efficiency of MobileNet and how it would perform against slower, deeper models.

DenseNet121 uses dense blocks and connectivity to create a model, Using these attributes in combination with bottleneck layers DenseNet121 is used to create a CNN that is able to generate models that have shown high levels of accuracy. The architecture for the DenseNet121 model is shown in Figure 3.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figure 3 - DenseNet121 Architecture (Huang et al., 2016, p.3)

As seen in Figure 3 this model contains various transition and dense layers to create a model for multi class classification. DenseNet differs from traditional CNNs as “To further improve the information flow between layers we propose a different connectivity pattern: we introduce direct connections from any layer to all subsequent layers.“ (Hunag G et al, 2016, p3). Interconnected dense layers improve the information obtained by the model and thus allows it to learn more. These dense blocks seen in figure 3 consist of many dense layers that are connected as mentioned above.

ResNet50 is a variant of the ResNet architecture used in image classification. ResNet50 contains 48 convolutional layers which is ideal for training very deep convolutional neural networks. It also contains one max pooling layer and one average pooling layer, which is what gives it the 50 in the ResNet50 name. The model architecture is seen in Figure 4.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4 - Resnet Architecture (He et al., 2015, p.5)

There are other versions of ResNet however we decided to use ResNet50 to train our model and evaluate the results. ResNet networks were developed to solve the issue of how when a model gets “too deep” meaning too many layers with too many units the model tends to perform worse. This is called vanishing/exploding gradients. ResNet challenges this by creating a deep model that does not degrade the deeper the model gets. In (He et al, 2015, p1) it is stated that “There exists a solution by construction to the deeper model: the added layers are identity mapping, and the other layers are copied from the learned shallower model” As seen in Figure 4, the more layers in the ResNet model increase the overall accuracy, however the model tends to learn more slowly. ResNets allow for gradients to flow from later layers to the initial filters allowing for better classification.

The final CNN we chose to test was MobileNetV2. MobileNetV2, as the name suggests, is very similar to MobileNet but has a significantly lower parameter count. MobileNetV2, similarly to MobileNet is a very lightweight CNN with the following layers:

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 5 - MobileNetV2 Architecture (Sandler et al., 2018, p.5)

MobileNetV2 is designed to be lightweight, and to be used for mobile applications and other situations when resources may be lower than what is used for traditional deep learning models. MobileNetv2 uses a combination of depthwise separable convolutions, linear bottlenecks and inverted residuals. Despite this being a lightweight model it still yields good results because of the fact that “This module takes as an input a low-dimensional compressed representation which is first expanded to high dimension and filtered with a lightweight depthwise convolution. Features are subsequently projected back to a low-dimensional representation with a linear convolution” (Sandler et al., 2018, p.1) allowing for this lightweight model to provide comparable results to its more heavyweight counterparts.

Results

Custom Model:

Train: loss: 0.7791 - accuracy: 0.7633 - val_loss: 1.2727 - val_accuracy: 0.6925
Test: loss: 1.6618 - accuracy: 0.5395

	precision	recall	f1-score	support
0	0.73	0.50	0.59	38
1	0.47	0.47	0.47	38
2	0.57	0.64	0.60	45
3	0.50	0.51	0.51	43
4	0.65	0.81	0.72	57
5	0.60	0.76	0.67	46
6	0.46	0.21	0.29	52
7	0.50	0.51	0.51	41
8	0.70	0.76	0.73	55
9	0.31	0.33	0.32	40
10	0.84	0.85	0.84	54
11	0.46	0.61	0.52	56
12	0.58	0.40	0.48	62
13	0.38	0.57	0.46	54
14	0.35	0.30	0.32	47
15	0.46	0.24	0.32	45
accuracy			0.54	773
macro avg	0.54	0.53	0.52	773
weighted avg	0.54	0.54	0.53	773

Runtime on tensorflow GPU: 20 minutes 47.8 seconds

Figure 6: Custom ModelAccuracy and Loss

MobileNet:

Train: loss: 0.7608 - accuracy: 0.7577 - val_loss: 0.6890 - val_accuracy: 0.8050
Test: loss: 1.0851 - accuracy: 0.6831

	precision	recall	f1-score	support
0	0.79	0.68	0.73	38
1	0.66	0.76	0.71	38
2	0.76	0.69	0.72	45
3	0.58	0.60	0.59	43
4	0.66	0.77	0.71	57
5	0.76	0.85	0.80	46
6	0.64	0.40	0.49	52
7	0.51	0.68	0.58	41
8	0.76	0.91	0.83	55
9	0.57	0.50	0.53	40
10	0.94	0.83	0.88	54
11	0.65	0.62	0.64	56
12	0.57	0.71	0.63	62
13	0.82	0.76	0.79	54
14	0.63	0.51	0.56	47
15	0.69	0.56	0.62	45
accuracy			0.68	773
macro avg	0.69	0.68	0.68	773
weighted avg	0.69	0.68	0.68	773

Runtime on tensorflow GPU: 21 minutes 23.0 seconds

Figure 7: MobileNet Accuracy and Loss

DenseNet121:

Train: loss: 0.9721 accuracy: 0.6509 - val_loss: 0.9355 - val_accuracy: 0.7075
Test: loss: 1.6380 - accuracy: 0.5821

	precision	recall	f1-score	support
0	0.76	0.42	0.54	38
1	0.68	0.39	0.50	38
2	0.56	0.76	0.64	45
3	0.48	0.51	0.49	43
4	0.69	0.75	0.72	57
5	0.55	0.76	0.64	46
6	0.45	0.40	0.42	52
7	0.65	0.49	0.56	41
8	0.83	0.78	0.80	55
9	0.54	0.38	0.44	40
10	0.90	0.81	0.85	54
11	0.50	0.46	0.48	56
12	0.58	0.68	0.62	62
13	0.45	0.72	0.55	54
14	0.44	0.36	0.40	47
15	0.46	0.40	0.43	45
accuracy			0.58	773
macro avg	0.59	0.57	0.57	773
weighted avg	0.59	0.58	0.58	773

Runtime on tensorflow GPU: 33 minutes 44.7 seconds

Figure 8: DenseNet121 Accuracy and Loss

ResNet50:

Train: loss: 1.4360 - accuracy: 0.5477 - val_loss: 1.2312 - val_accuracy: 0.6075
Test: loss: 1.6710 - accuracy: 0.4864

	precision	recall	f1-score	support
0	0.62	0.42	0.50	38
1	0.75	0.24	0.36	38
2	0.49	0.71	0.58	45
3	0.53	0.42	0.47	43
4	0.63	0.75	0.69	57
5	0.56	0.59	0.57	46
6	0.23	0.31	0.26	52
7	0.53	0.44	0.48	41
8	0.75	0.69	0.72	55
9	0.32	0.30	0.31	40
10	0.89	0.76	0.82	54
11	0.42	0.41	0.41	56
12	0.58	0.47	0.52	62
13	0.37	0.43	0.39	54
14	0.32	0.13	0.18	47
15	0.27	0.56	0.36	45
accuracy			0.49	773
macro avg	0.52	0.48	0.48	773
weighted avg	0.52	0.49	0.49	773

Figure 9: ResNet50 Accuracy and Loss

Runtime on tensorflow GPU:
23 minutes 31.4 seconds

MobileNetV2:

Train: loss: 1.3171 - accuracy: 0.5787 - val_loss: 1.3648 - val_accuracy: 0.5925
Test: loss: 1.7139 - accuracy: 0.4618

	precision	recall	f1-score	support
0	0.75	0.24	0.36	38
1	0.35	0.53	0.42	38
2	0.77	0.44	0.56	45
3	0.51	0.51	0.51	43
4	0.48	0.75	0.59	57
5	0.76	0.48	0.59	46
6	0.42	0.25	0.31	52
7	0.30	0.68	0.42	41
8	0.63	0.71	0.67	55
9	0.67	0.05	0.09	40
10	0.96	0.81	0.88	54
11	0.52	0.27	0.35	56
12	0.50	0.35	0.42	62
13	0.25	0.72	0.37	54
14	1.00	0.02	0.04	47
15	0.34	0.40	0.37	45
accuracy			0.46	773
macro avg	0.58	0.45	0.43	773
weighted avg	0.57	0.46	0.44	773

Runtime on tensorflow GPU: 22 minutes 15.2 seconds

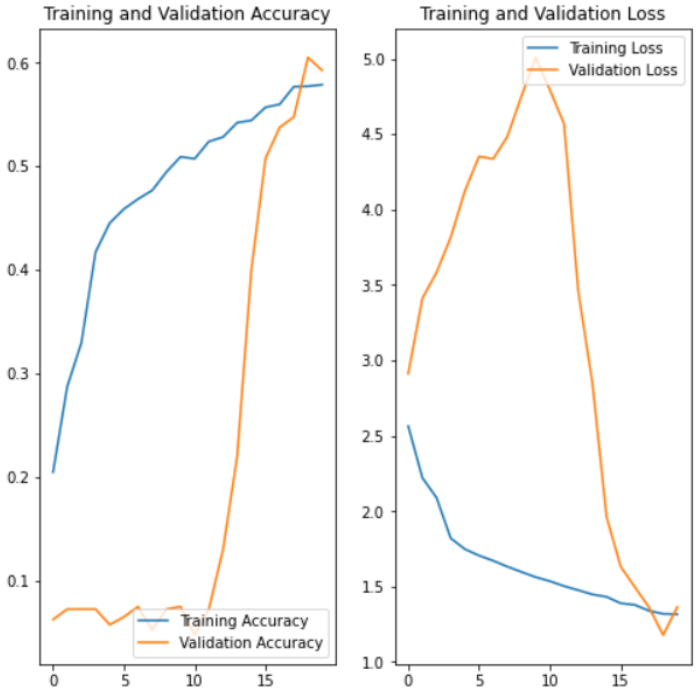

Figure 10: MobileNetV2 Accuracy and Loss

Table 1 : Final Test Results

Model	Time to Fit	Test Accuracy	Precision (Macro Average)	Recall (Macro Average)	F1-Score (Macro Average)	Overall Ranking
MobileNet	21min 23.0s	0.6831	0.69	0.68	0.68	1
DenseNet 121	33min 44.7s	0.5821	0.59	0.57	0.57	2
Custom Model	20min 47.8s	0.5395	0.54	0.53	0.52	3
ResNet50	23min 31.4s	0.4864	0.52	0.48	0.48	4

MobileNet V2	22min 15.2s	0.4618	0.58	0.45	0.43	5

Table 2: Average F1-Score by Class (All Models)

Class	Average F1-Score
0 - Baseball	0.544
1 - Basketball	0.492
2 - Boxing	0.62
3 - Fencing	0.514
4 - Football (Soccer)	0.686
5 - Formula1	0.654
6 - Gymnastics	0.354
7 - Field Hockey	0.51
8 - Ice Hockey	0.75
9 - Shooting	0.338
10 - Swimming	0.854
11 - Table Tennis	0.48
12 - Tennis	0.534
13 - Volleyball	0.512
14 - Weight-lifting	0.30
15 - Wrestling	0.42

Discussion

The results show that MobileNet was the most superior algorithm during our testing. All the models were tested in the same way. They were given the same sports image dataset with 16 classes, set up with identical hyperparameters and run for the same amount of time. MobileNet beat all other models in accuracy, precision and recall. It did this while having the second fastest time to train. MobileNet's unique design of

convolutional layers secured its position as the best model tested. During testing we expected the models DenseNet121 and ResNet50 to perform better than MobileNet as the former two are more complex models. MobileNet was built for less powerful hardware such as mobile devices and embedded vision applications. (Howard et al., 2017, p.1).

Our original thoughts on model performance were based on depth, parameters and its architecture. Specifically, our team originally thought DenseNet121 would be the strongest overall model. The ability for DenseNet to utilize feature-maps from proceeding layers should make this model more accurate and compact (Huang et al., 2016, p.8). However, during our experiments, DenseNet121 only ranked second in accuracy, precision and recall. It also took the longest amount of time to train.

The custom CNN model created by our team performed better than some of the other researchers models. This model followed the standard CNN approach, mixing convolutional layers, pooling layer and dense layer to build the model. This model had the fastest time to train. It placed third in our model ranking.

In addition to the models' performances we looked at the classification performance of each sport. The models performed best on average with swimming, ice hockey and football (soccer). The worst average classification results were with weight-lifting, gymnastics and shooting. Every model tested had excellent performance in classifying swimming images. This is most likely due to swimming being the only aquatic sport. So, it was the only class of images that would include water and swimming pools. The average F1-score for swimming was 0.854. No other sport had a strong performance like that across all the models. The worst performing sport was weight-lifting. This sport had an average F1-score of 0.30 across all the models. This poor performance could be attributed to the ambiguity of the sport. There are many different weight-lifting styles and movements, such as bench press, squatting, deadlifting, clean and jerk, and more. The images featured multiple kinds of equipment like dumbbells, barbells and workout machines.

Reflecting on our testing, we are happy with the results. If we were to repeat this experiment and do anything differently it would be to train the models for additional epochs. 20 epochs, with a batch size of 16 and steps per epoch of 466 was plenty of processing time. We were concerned that additional training would lead to overfitting. If the dataset was expanded and included more images, we would have considered more training epochs. There is a possibility that some of the more complex models would benefit from additional training epochs. However, we concluded that our training time was sufficient to compare the results from the different models.

Conclusion

This report discusses the architecture and performance of several convolutional neural network models. The models tested were DenseNet121, MobileNet,

MobileNetV2, ResNet50 and a custom model. The models were trained and tested with a Sports image dataset that included 10094 unique images divided into 16 classes of different sports. All models were given the same hyperparameters and trained for the same amount of epochs. The experiment showed how each of the models performed with the given data. We can conclude that with the dataset used, MobileNet provided the best results even though it was not the most complex model. In future research projects related to sports images classification, we would expect a larger dataset and training the model through more epochs to provide more accurate results. With these changes made to a future project, we would predict that more complex models like DenseNet121 and ResNet50 would outperform the other models.

References

[1] He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition*. ArXiv.org. Retrieved April 17, 2022, from https://arxiv.org/abs/1512.03385

[2] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. Google Inc. Retrieved April 14, 2022, https://arxiv.org/abs/1704.04861

[3] Huang, G., Liu, Z., & Weinberger, Kilian Q. (2016). *Densely Connected Convolutional Networks.* ArXiv.org. Retrieved April 16, 2022, from https://arxiv.org/abs/1608.06993

[4] *Machine Learning Glossary*. (n.d.). Google Developers. Retrieved April 9, 2022, from https://developers.google.com/machine-learning/glossary/#neural_network

[5] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks.* ArXiv.org. Retrieved April 15, 2022, https://arxiv.org/abs/1801.04381

[6] Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. ArXiv.org. Retrieved April 18, 2022, from https://arxiv.org/abs/1409.1556

[7] *Sports Image Dataset*. (n.d.). Www.kaggle.com. Retrieved April 8, 2022, from https://www.kaggle.com/datasets/sovitrath/sports-image-dataset

[8] Team, K. (n.d.). *Keras documentation: Keras Applications*. Keras.io. Retrieved April 10, 2022, from https://keras.io/api/applications/#usage-examples-for-image-classification-models