**import** os  
**import** mediapipe **as** mp   
**import** numpy **as** np  
**import** cv2  
**import** math  
**import** pickle  
**import** tensorflow **as** tf  
**import** matplotlib.pyplot **as** plt

[24]

list\_of\_metrics = {}  
list\_of\_metrics\_gesture = {}

**Common Functions used:**

[10]

**def** preprocess(data, labels):  
      
    **import** tensorflow **as** tf  
    **from** sklearn.model\_selection **import** train\_test\_split  
  
    *# STEP 1: Data Distribution Check*  
    print(**"Data Shape: "**, data.shape, **"||  Label Shape: "**, labels.shape)  
    unique, counts = np.unique(labels, return\_counts=**True**)  
    print(**"\nOriginal Label Distribution:\n"**, dict(zip(unique, counts)))  
  
    *# STEP 2: Convert labels to a Categorical Variable*  
    lab = tf.keras.utils.to\_categorical(labels)  
  
    *# STEP 3: Train-Test Split*  
    X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, lab, test\_size=0.2, random\_state=42)  
  
    unique, counts = np.unique(np.argmax(y\_train, axis=1), return\_counts=**True**)  
    print(**"\nTraining:\n"**, X\_train.shape, y\_train.shape, **"\n"**) *#, dict(zip(unique, counts)))*  
  
    unique, counts = np.unique(np.argmax(y\_test, axis=1), return\_counts=**True**)  
    print(**"Test:\n"**, X\_test.shape, y\_test.shape, **"\n"**) *#, dict(zip(unique, counts)))*  
  
    **return** X\_train, X\_test, y\_train, y\_test

[11]

**def** train\_model(input\_shape = (21,2), output\_shape=24):  
    **import** tensorflow **as** tf  
    **from** tensorflow.keras.models **import** Sequential  
    **from** tensorflow.keras.layers **import** Dense, Flatten, Dropout, LSTM  
    **from** tensorflow.keras.callbacks **import** ModelCheckpoint, EarlyStopping  
  
    **if**(input\_shape[1]==0):  
        model = Sequential([  
            Flatten(input\_shape=(input\_shape[0],)),  
            Dense(64,  activation=**'relu'**),  
            Dense(128, activation=**'relu'**),  
            Dense(512, activation=**'relu'**),  
            Dropout(rate=0.1),  
            Dense(512, activation=**'relu'**),  
            Dropout(rate=0.2),  
            Dense(64, activation=**'relu'**),  
            Dense(32, activation=**'relu'**),  
            Dense(output\_shape, activation=**'softmax'**),  
        ])  
    **else**:  
        model = Sequential([  
            Flatten(input\_shape=(input\_shape[0],input\_shape[1])),  
            Dense(64,  activation=**'relu'**),  
            Dense(128, activation=**'relu'**),  
            Dense(512, activation=**'relu'**),  
            Dropout(rate=0.1),  
            Dense(512, activation=**'relu'**),  
            Dropout(rate=0.2),  
            Dense(64, activation=**'relu'**),  
            Dense(32, activation=**'relu'**),  
            Dense(output\_shape, activation=**'softmax'**),  
        ])  
  
    *# model.summary()*  
  
    callback\_checkpoint = ModelCheckpoint(  
        **'./checkpoints/'**, monitor=**'val\_loss'**, verbose=0, save\_best\_only=**True**,  
        save\_weights\_only=**True**, mode=**'auto'**, save\_freq=**'epoch'**,  
    )  
    callback\_loss = EarlyStopping(monitor=**'loss'**, patience=4)  
  
    model.compile(loss=**'categorical\_crossentropy'**, optimizer=**'adam'**, metrics=[**'accuracy'**])  
  
    history = model.fit(X\_train, y\_train,   
                        epochs=50,   
                        callbacks=[callback\_checkpoint, callback\_loss])  
    **return** model, history

[12]

**def** evaluate\_model(history, X\_test, y\_test, model, title, actions):  
  **"""**  
**Input:**  
**history - Model history variable**  
**X\_test - Test features**  
**y\_test - Target test variables**  
**model - Keras Model**  
  
**Output:**  
**Accuracy Plot (Training + Validation)**  
**Loss Plot (Training + Validation)**  
**Confusion matrix**  
**Accuracy**  
**Classification Report**  
**"""**    
  
  **from** sklearn.metrics **import** classification\_report, confusion\_matrix  
  **import** matplotlib.pyplot **as** plt  
  **import** seaborn **as** sns  
    
  scores = model.evaluate((X\_test),y\_test, verbose=0)  
    
  fig, axs = plt.subplots(1,3, figsize=(35,8))  
  fig.tight\_layout()  
  
  axs[0].plot(history.history[**'accuracy'**])  
  axs[0].set\_xlabel(**'Epoch'**)  
  axs[0].set\_ylabel(**'Accuracy'**)  
  axs[0].set\_title(**'Model - Accuracy ('** + title +**")"**)  
  axs[0].legend([**'Training'**], loc=**'lower right'**)  
  
  axs[1].plot(history.history[**'loss'**])  
  axs[1].set\_xlabel(**'Epoch'**)  
  axs[1].set\_ylabel(**'Model Loss'**)  
  axs[1].set\_title(**'Model - Loss ('** + title +**")"**)  
  axs[1].legend([**'Training'**], loc=**'upper right'**)  
  
  target\_names=actions  
    
  y\_true=[]  
  **for** element **in** y\_test:  
      y\_true.append(np.argmax(element))  
  
  prediction\_proba = model.predict(X\_test)  
  prediction = np.argmax(prediction\_proba,axis=1)  
  cnf\_matrix = confusion\_matrix(y\_true, prediction)      
  g = sns.heatmap(cnf\_matrix, annot=**True**, cmap=**'Blues'**, fmt=**'g'**, ax=axs[2],  
                  xticklabels=target\_names, yticklabels=target\_names)  
  g.set\_yticklabels(g.get\_yticklabels(), rotation = 60)  
  axs[2].set\_title(**"Confusion Matrix"**)  
  plt.show()  
  
  
  print(**"\nAccuracy: %.2f%% \n"** % (scores[1]\*100))  
  print(classification\_report(y\_true, prediction, target\_names=target\_names))  
  
  **return** {**"metrics"**: classification\_report(y\_true, prediction, target\_names=target\_names, output\_dict=**True**)}

[13]

**def** getAngle(a, b, c):  
    ang = math.degrees(math.atan2(c[1]-b[1], c[0]-b[0]) - math.atan2(a[1]-b[1], a[0]-b[0]))  
    **return** ang + 360 **if** ang < 0 **else** ang

[14]

**def** get\_distance(a,b):  
    **from** math **import** pow, sqrt  
    **return** round(sqrt(pow(b[0]-a[0],2) + pow(b[1]-a[1],2)), 4)

**1. Alphabet Landmarks**

**Loading Landmarks and Labels**

[16]

LANDMARK\_PATH = os.path.join(**'./data/Character\_Data\_Landmarks'**)  
  
actions = [chr(i) **for** i **in** range(ord(**'A'**), ord(**'Z'**)) **if** chr(i)!=**'T'**]  
  
no\_sequences =  25  
sequence\_length = 30  
  
print(actions)  
  
label\_map = {label:num **for** num, label **in** enumerate(actions)}  
print(label\_map)

['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'U', 'V', 'W', 'X', 'Y']  
{'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8, 'J': 9, 'K': 10, 'L': 11, 'M': 12, 'N': 13, 'O': 14, 'P': 15, 'Q': 16, 'R': 17, 'S': 18, 'U': 19, 'V': 20, 'W': 21, 'X': 22, 'Y': 23}

**A. Neural Networks**

**i) Just (X,Y)**

[90]

missing = []  
data = []  
labels = []  
  
**with** open(**'data/dimensions.pkl'**, **'rb'**) **as** f:  
    x, y, c = pickle.load(f)  
  
**for** action **in** actions:  
    *# Loop through sequences aka videos*  
    **for** sequence **in** range(0, 0 + no\_sequences):   
        *# Loop through video length aka sequence length*  
        **for** frame\_num **in** range(sequence\_length):  
              
            pkl\_path = os.path.join(LANDMARK\_PATH, action, str(sequence), str(frame\_num))  
            **try**:  
                **with** open(pkl\_path, **'rb'**) **as** f:  
                    result = pickle.load(f)  
  
                landmarks = []  
                **for** handslms **in** result:  
                    **for** lm **in** handslms.landmark:  
                        lmx = int(lm.x \* x)  
                        lmy = int(lm.y \* y)  
                          
                        *# landmarks.append([lmx, lmy])*  
                        landmarks.append(lmy)  
                        landmarks.append(lmy)  
                  
                data.append(landmarks)  
                labels.append(label\_map[action])  
                  
            **except**:  
                missing.append(pkl\_path)  
  
              
data = np.array(data)  
labels = np.array(labels)

[91]

print(data.shape, labels.shape)

(17764, 42) (17764,)

[92]

X\_train, X\_test, y\_train, y\_test = preprocess(data, labels)

Data Shape: (17764, 42) || Label Shape: (17764,)  
  
Original Label Distribution:  
 {0: 747, 1: 748, 2: 746, 3: 748, 4: 749, 5: 749, 6: 605, 7: 718, 8: 748, 9: 746, 10: 749, 11: 749, 12: 750, 13: 749, 14: 749, 15: 748, 16: 731, 17: 748, 18: 750, 19: 745, 20: 746, 21: 748, 22: 749, 23: 749}  
  
Training:  
 (14211, 42) (14211, 24)   
  
Test:  
 (3553, 42) (3553, 24)

[93]

model\_xy, history = train\_model( (X\_train.shape[1],0), y\_train.shape[1])  
*# model\_xy, history = train\_model( (X\_train.shape[1], X\_train.shape[2]), y\_train.shape[1])*

Epoch 1/50  
445/445 [==============================] - 2s 3ms/step - loss: 2.2048 - accuracy: 0.4910  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 2/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.5899 - accuracy: 0.7662  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 3/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.4880 - accuracy: 0.8151  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 4/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.3923 - accuracy: 0.8493  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 5/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.4156 - accuracy: 0.8463  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 6/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.3457 - accuracy: 0.8726  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 7/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.3677 - accuracy: 0.8608  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 8/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.3374 - accuracy: 0.8745  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 9/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.3699 - accuracy: 0.8677  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 10/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.3130 - accuracy: 0.8842  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 11/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.3036 - accuracy: 0.8872  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 12/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.3010 - accuracy: 0.8930  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 13/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.3266 - accuracy: 0.8825  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 14/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.2973 - accuracy: 0.8900  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 15/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.2772 - accuracy: 0.8960  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 16/50  
445/445 [==============================] - 1s 3ms/step - loss: 0.2896 - accuracy: 0.8961  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 17/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.2694 - accuracy: 0.9044  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 18/50  
445/445 [==============================] - 1s 3ms/step - loss: 0.5764 - accuracy: 0.7970  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 19/50  
445/445 [==============================] - 1s 3ms/step - loss: 0.2747 - accuracy: 0.9045  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 20/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.2776 - accuracy: 0.8994  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 21/50  
445/445 [==============================] - 1s 3ms/step - loss: 0.2491 - accuracy: 0.9150  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 22/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.2836 - accuracy: 0.8999  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 23/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.2497 - accuracy: 0.9085  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 24/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.2775 - accuracy: 0.9023  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 25/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.2718 - accuracy: 0.9048  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.

[75]

model\_xy.save(**"./model\_saves/only\_coordinates"**)

INFO:tensorflow:Assets written to: ./model\_saves/only\_coordinates\assets

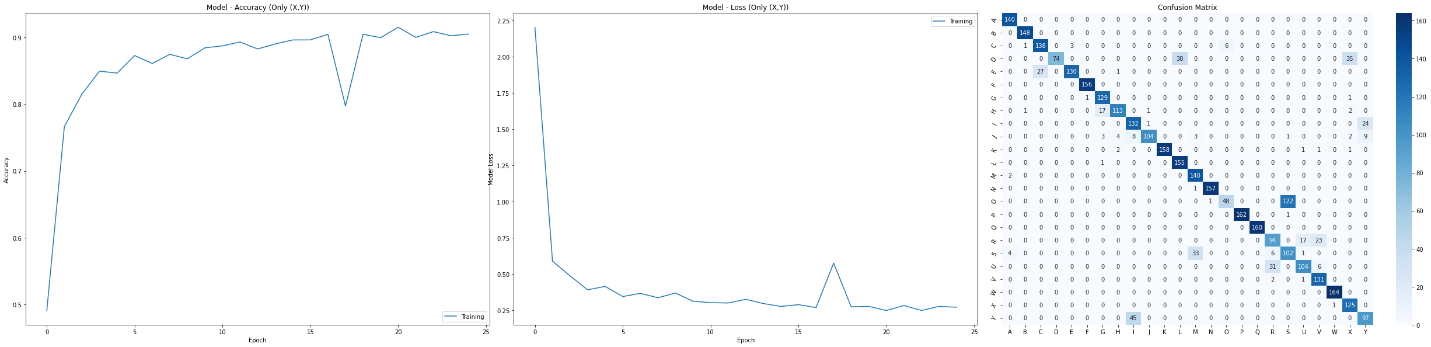
[94]

**from** tensorflow.keras.models **import** load\_model  
  
model\_xy = load\_model(**"./model\_saves/only\_coordinates"**)

[95]

only\_coordinates\_results = evaluate\_model(history, X\_test, y\_test, model\_xy, **"Only (X,Y)"**, actions)

Accuracy: 86.15%   
  
 precision recall f1-score support  
  
 A 0.96 1.00 0.98 140  
 B 0.99 1.00 0.99 148  
 C 0.84 0.93 0.88 148  
 D 1.00 0.50 0.67 147  
 E 0.98 0.82 0.89 158  
 F 0.99 1.00 1.00 156  
 G 0.86 0.98 0.92 131  
 H 0.94 0.84 0.89 134  
 I 0.71 0.84 0.77 157  
 J 0.98 0.78 0.87 134  
 K 1.00 0.97 0.98 163  
 L 0.80 0.99 0.89 156  
 M 0.79 0.99 0.88 142  
 N 0.99 0.99 0.99 158  
 O 0.89 0.28 0.43 171  
 P 1.00 0.99 1.00 163  
 Q 1.00 1.00 1.00 160  
 R 0.71 0.70 0.70 134  
 S 0.45 0.70 0.55 146  
 U 0.84 0.74 0.78 141  
 V 0.81 0.98 0.89 134  
 W 0.99 1.00 1.00 164  
 X 0.75 0.99 0.86 126  
 Y 0.75 0.68 0.71 142  
  
 accuracy 0.86 3553  
 macro avg 0.88 0.86 0.86 3553  
weighted avg 0.88 0.86 0.86 3553



[20]

*# with open('data/results\_new.pkl', 'rb') as f:*  
*#     list\_of\_metrics = pickle.load(f)*  
  
list\_of\_metrics[**'only\_coordinates'**] = only\_coordinates\_results  
  
**with** open(**'data/results\_new.pkl'**, **'wb'**) **as** f:  
     pickle.dump(list\_of\_metrics, f)

[22]

**with** open(**'data/results\_new.pkl'**, **'rb'**) **as** f:  
    list\_of\_metrics = pickle.load(f)  
  
**for** key **in** list\_of\_metrics.keys():  
    print(key)

only\_coordinates

**import** tensorflowjs **as** tfjs  
  
tfjs.converters.save\_keras\_model(model\_xy, **"./public"**)

**ii) Two Angles**

[17]

missing = []  
data = []  
labels = []  
mpHands = mp.solutions.hands  
  
**with** open(**'data/dimensions.pkl'**, **'rb'**) **as** f:  
    x, y, c = pickle.load(f)  
  
**for** action **in** actions:  
    *# Loop through sequences aka videos*  
    **for** sequence **in** range(0, 0 + no\_sequences):   
        *# Loop through video length aka sequence length*  
        **for** frame\_num **in** range(sequence\_length):  
              
            pkl\_path = os.path.join(LANDMARK\_PATH, action, str(sequence), str(frame\_num))  
            **try**:  
                **with** open(pkl\_path, **'rb'**) **as** f:  
                    result = pickle.load(f)  
  
                *# print(pkl\_path)*  
                landmarks = []  
                **for** handslms **in** result:  
                    angle\_1 = getAngle((handslms.landmark[mpHands.HandLandmark.THUMB\_TIP].x, handslms.landmark[mpHands.HandLandmark.THUMB\_TIP].y),  
                                            (handslms.landmark[mpHands.HandLandmark.WRIST].x, handslms.landmark[mpHands.HandLandmark.WRIST].y),  
                                            (handslms.landmark[mpHands.HandLandmark.PINKY\_TIP].x, handslms.landmark[mpHands.HandLandmark.PINKY\_TIP].y))  
  
                    angle\_2 = getAngle((handslms.landmark[mpHands.HandLandmark.INDEX\_FINGER\_TIP].x, handslms.landmark[mpHands.HandLandmark.INDEX\_FINGER\_TIP].y),  
                                        (handslms.landmark[mpHands.HandLandmark.INDEX\_FINGER\_MCP].x, handslms.landmark[mpHands.HandLandmark.INDEX\_FINGER\_MCP].y),  
                                        (handslms.landmark[mpHands.HandLandmark.MIDDLE\_FINGER\_TIP].x, handslms.landmark[mpHands.HandLandmark.MIDDLE\_FINGER\_TIP].y))  
                    *# print(angle\_1, angle\_2)*  
                    **for** lm **in** handslms.landmark:  
                        lmx = int(lm.x \* x)  
                        lmy = int(lm.y \* y)  
                          
                        landmarks.append([lmx, lmy])  
                        *# landmarks.append(lmx)*  
                        *# landmarks.append(lmy)*  
                    landmarks.append([angle\_1, angle\_2])  
                    *# landmarks.append(angle\_1)*  
                    *# landmarks.append(angle\_2)*  
  
                data.append(landmarks)  
                labels.append(label\_map[action])  
                  
            **except**:  
                missing.append(pkl\_path)  
  
              
data = np.array(data)  
labels = np.array(labels)

[18]

print(data.shape, labels.shape)

(17764, 22, 2) (17764,)

[19]

X\_train, X\_test, y\_train, y\_test = preprocess(data, labels)

Data Shape: (17764, 22, 2) || Label Shape: (17764,)  
  
Original Label Distribution:  
 {0: 747, 1: 748, 2: 746, 3: 748, 4: 749, 5: 749, 6: 605, 7: 718, 8: 748, 9: 746, 10: 749, 11: 749, 12: 750, 13: 749, 14: 749, 15: 748, 16: 731, 17: 748, 18: 750, 19: 745, 20: 746, 21: 748, 22: 749, 23: 749}  
  
Training:  
 (14211, 22, 2) (14211, 24)   
  
Test:  
 (3553, 22, 2) (3553, 24)

[21]

*# model\_two\_angles, history = train\_model( (X\_train.shape[1], 0), y\_train.shape[1])*  
model\_two\_angles, history = train\_model( (X\_train.shape[1], X\_train.shape[2]), y\_train.shape[1])

Epoch 1/50  
445/445 [==============================] - 2s 4ms/step - loss: 3.3380 - accuracy: 0.1236  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 2/50  
445/445 [==============================] - 2s 4ms/step - loss: 1.2937 - accuracy: 0.6037  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 3/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.4504 - accuracy: 0.8424  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 4/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.2687 - accuracy: 0.9075  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 5/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.2876 - accuracy: 0.9059  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 6/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.2344 - accuracy: 0.9223  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 7/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1700 - accuracy: 0.9459  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 8/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1896 - accuracy: 0.9436  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 9/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.1922 - accuracy: 0.9398  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 10/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.1484 - accuracy: 0.9548  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 11/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.1759 - accuracy: 0.9502  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 12/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1219 - accuracy: 0.9637  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 13/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.1275 - accuracy: 0.9653  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 14/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.1760 - accuracy: 0.9469  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 15/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1976 - accuracy: 0.9443  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 16/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.1128 - accuracy: 0.9684  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 17/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1340 - accuracy: 0.9636  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 18/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.1382 - accuracy: 0.9615  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 19/50  
445/445 [==============================] - 2s 3ms/step - loss: 0.2033 - accuracy: 0.9424  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 20/50  
445/445 [==============================] - 1s 3ms/step - loss: 0.1175 - accuracy: 0.9688  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.

[23]

model\_two\_angles.save(**'./model\_saves/two\_angles\_new'**)

INFO:tensorflow:Assets written to: ./model\_saves/two\_angles\_new\assets

[5]

**from** tensorflow.keras.models **import** load\_model  
  
model\_two\_angles = load\_model(**'./model\_saves/two\_angles'**)

[24]

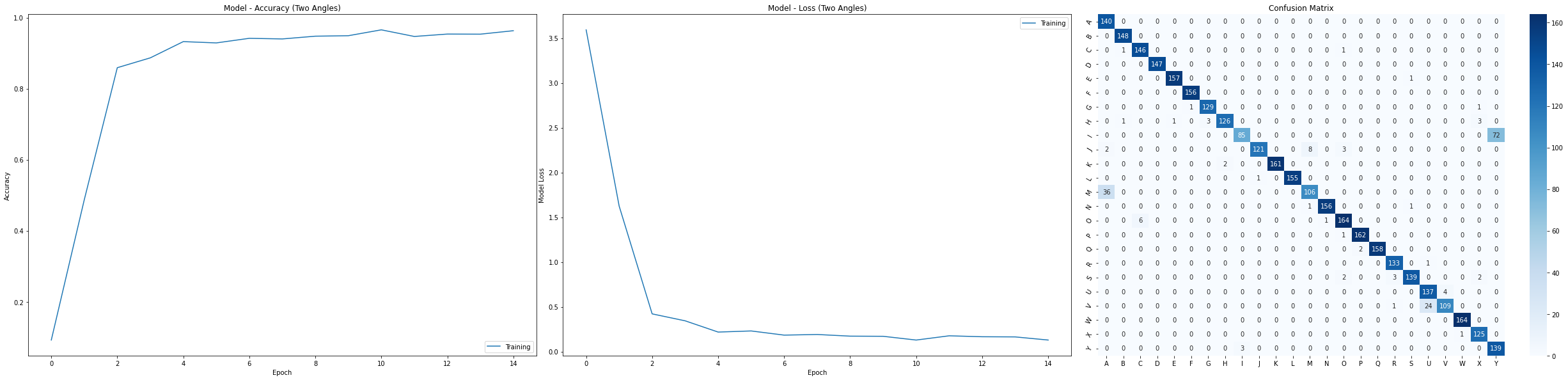
model\_two\_angles.summary()

Model: "sequential\_1"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
flatten\_1 (Flatten) (None, 44) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_7 (Dense) (None, 64) 2880   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_8 (Dense) (None, 128) 8320   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_9 (Dense) (None, 512) 66048   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout\_2 (Dropout) (None, 512) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_10 (Dense) (None, 512) 262656   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout\_3 (Dropout) (None, 512) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_11 (Dense) (None, 64) 32832   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_12 (Dense) (None, 32) 2080   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_13 (Dense) (None, 24) 792   
=================================================================  
Total params: 375,608  
Trainable params: 375,608  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[112]

two\_angles\_results = evaluate\_model(history, X\_test, y\_test, model\_two\_angles, **"Two Angles"**, actions)

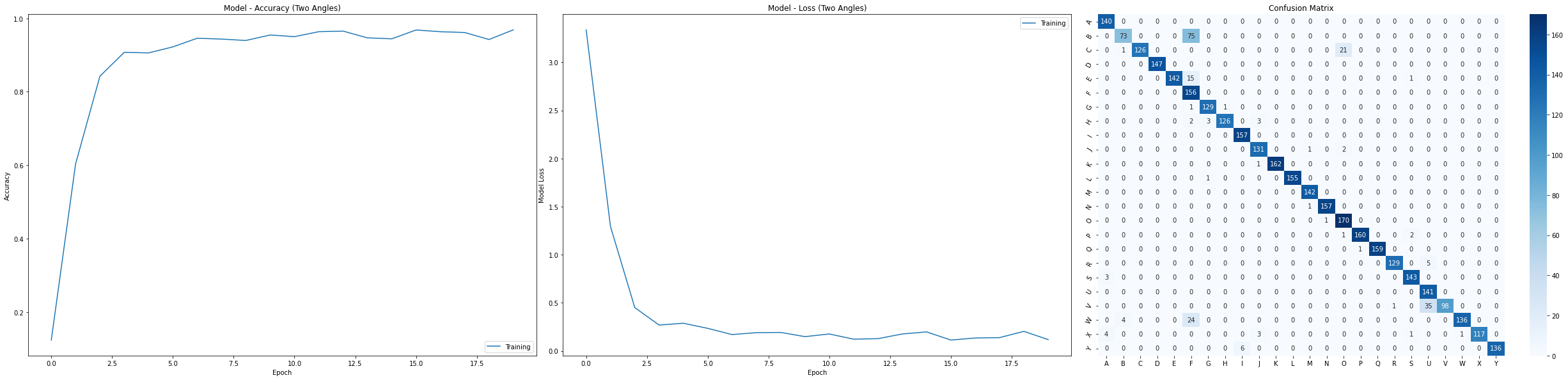
Accuracy: 94.65%   
  
 precision recall f1-score support  
  
 A 0.79 1.00 0.88 140  
 B 0.99 1.00 0.99 148  
 C 0.96 0.99 0.97 148  
 D 1.00 1.00 1.00 147  
 E 0.99 0.99 0.99 158  
 F 0.99 1.00 1.00 156  
 G 0.98 0.98 0.98 131  
 H 0.98 0.94 0.96 134  
 I 0.97 0.54 0.69 157  
 J 0.99 0.90 0.95 134  
 K 1.00 0.99 0.99 163  
 L 1.00 0.99 1.00 156  
 M 0.92 0.75 0.82 142  
 N 0.99 0.99 0.99 158  
 O 0.96 0.96 0.96 171  
 P 0.99 0.99 0.99 163  
 Q 1.00 0.99 0.99 160  
 R 0.97 0.99 0.98 134  
 S 0.99 0.95 0.97 146  
 U 0.85 0.97 0.90 141  
 V 0.96 0.81 0.88 134  
 W 0.99 1.00 1.00 164  
 X 0.95 0.99 0.97 126  
 Y 0.66 0.98 0.79 142  
  
 accuracy 0.95 3553  
 macro avg 0.95 0.95 0.94 3553  
weighted avg 0.95 0.95 0.95 3553



[22]

two\_angles\_results = evaluate\_model(history, X\_test, y\_test, model\_two\_angles, **"Two Angles"**, actions)

Accuracy: 93.78%   
  
 precision recall f1-score support  
  
 A 0.95 1.00 0.98 140  
 B 0.94 0.49 0.65 148  
 C 1.00 0.85 0.92 148  
 D 1.00 1.00 1.00 147  
 E 1.00 0.90 0.95 158  
 F 0.57 1.00 0.73 156  
 G 0.97 0.98 0.98 131  
 H 0.99 0.94 0.97 134  
 I 0.96 1.00 0.98 157  
 J 0.95 0.98 0.96 134  
 K 1.00 0.99 1.00 163  
 L 1.00 0.99 1.00 156  
 M 0.99 1.00 0.99 142  
 N 0.99 0.99 0.99 158  
 O 0.88 0.99 0.93 171  
 P 0.99 0.98 0.99 163  
 Q 1.00 0.99 1.00 160  
 R 0.99 0.96 0.98 134  
 S 0.97 0.98 0.98 146  
 U 0.78 1.00 0.88 141  
 V 1.00 0.73 0.84 134  
 W 0.99 0.83 0.90 164  
 X 1.00 0.93 0.96 126  
 Y 1.00 0.96 0.98 142  
  
 accuracy 0.94 3553  
 macro avg 0.96 0.94 0.94 3553  
weighted avg 0.95 0.94 0.94 3553



[38]

list\_of\_metrics[**'two\_angles'**] = two\_angles\_results

[39]

**with** open(**'data/results\_new.pkl'**,**'wb'**) **as** f:  
    pickle.dump(list\_of\_metrics, f)

[25]

**import** tensorflowjs **as** tfjs  
  
tfjs.converters.save\_keras\_model(model\_two\_angles, **"../public/models/alphabet\_two\_angles"**)

**iii) Distance + Angles**

[131]

missing = []  
data = []  
labels = []  
mpHands = mp.solutions.hands  
  
**with** open(**'data/dimensions.pkl'**, **'rb'**) **as** f:  
    x, y, c = pickle.load(f)  
  
**for** action **in** actions:  
    *# Loop through sequences aka videos*  
    **for** sequence **in** range(0, 0 + no\_sequences):   
        *# Loop through video length aka sequence length*  
        **for** frame\_num **in** range(sequence\_length):  
              
            pkl\_path = os.path.join(LANDMARK\_PATH, action, str(sequence), str(frame\_num))  
            **try**:  
                **with** open(pkl\_path, **'rb'**) **as** f:  
                    result = pickle.load(f)  
  
                *# print(pkl\_path)*  
                landmarks = []  
                **for** handslms **in** result:  
                    angle\_1 = getAngle((handslms.landmark[mpHands.HandLandmark.THUMB\_TIP].x, handslms.landmark[mpHands.HandLandmark.THUMB\_TIP].y),  
                                            (handslms.landmark[mpHands.HandLandmark.WRIST].x, handslms.landmark[mpHands.HandLandmark.WRIST].y),  
                                            (handslms.landmark[mpHands.HandLandmark.PINKY\_TIP].x, handslms.landmark[mpHands.HandLandmark.PINKY\_TIP].y))  
  
                    angle\_2 = getAngle((handslms.landmark[mpHands.HandLandmark.INDEX\_FINGER\_TIP].x, handslms.landmark[mpHands.HandLandmark.INDEX\_FINGER\_TIP].y),  
                                        (handslms.landmark[mpHands.HandLandmark.INDEX\_FINGER\_MCP].x, handslms.landmark[mpHands.HandLandmark.INDEX\_FINGER\_MCP].y),  
                                        (handslms.landmark[mpHands.HandLandmark.MIDDLE\_FINGER\_TIP].x, handslms.landmark[mpHands.HandLandmark.MIDDLE\_FINGER\_TIP].y))  
                      
                      
                    distance\_1 = get\_distance( (handslms.landmark[mpHands.HandLandmark.THUMB\_TIP].x, handslms.landmark[mpHands.HandLandmark.THUMB\_TIP].y),  
                                                (handslms.landmark[mpHands.HandLandmark.INDEX\_FINGER\_TIP].x, handslms.landmark[mpHands.HandLandmark.INDEX\_FINGER\_TIP].y) )  
                      
                    distance\_2 = get\_distance( (handslms.landmark[mpHands.HandLandmark.THUMB\_TIP].x, handslms.landmark[mpHands.HandLandmark.THUMB\_TIP].y),  
                                                (handslms.landmark[mpHands.HandLandmark.PINKY\_TIP].x, handslms.landmark[mpHands.HandLandmark.PINKY\_TIP].y) )  
  
                    **for** lm **in** handslms.landmark:  
                        lmx = int(lm.x \* x)  
                        lmy = int(lm.y \* y)  
                          
                        landmarks.append([lmx, lmy])  
                        *# landmarks.append(lmx)*  
                        *# landmarks.append(lmy)*  
                    landmarks.append([angle\_1, angle\_2])  
                    landmarks.append([distance\_1, distance\_2])  
                    *# landmarks.append(angle\_1)*  
                    *# landmarks.append(angle\_2)*  
                    *# landmarks.append(distance\_1)*  
                    *# landmarks.append(distance\_2)*  
  
                data.append(landmarks)  
                labels.append(label\_map[action])  
                  
            **except**:  
                missing.append(pkl\_path)  
  
              
data = np.array(data)  
labels = np.array(labels)

[132]

print(data.shape, labels.shape)

(17764, 23, 2) (17764,)

[133]

X\_train, X\_test, y\_train, y\_test = preprocess(data, labels)

Data Shape: (17764, 23, 2) || Label Shape: (17764,)  
  
Original Label Distribution:  
 {0: 747, 1: 748, 2: 746, 3: 748, 4: 749, 5: 749, 6: 605, 7: 718, 8: 748, 9: 746, 10: 749, 11: 749, 12: 750, 13: 749, 14: 749, 15: 748, 16: 731, 17: 748, 18: 750, 19: 745, 20: 746, 21: 748, 22: 749, 23: 749}  
  
Training:  
 (14211, 23, 2) (14211, 24)   
  
Test:  
 (3553, 23, 2) (3553, 24)

[135]

model\_angle\_and\_distance, history = train\_model( (X\_train.shape[1], X\_train.shape[2]), y\_train.shape[1])  
*# model\_angle\_and\_distance, history = train\_model( (X\_train.shape[1], 0), y\_train.shape[1])*

Epoch 1/50  
445/445 [==============================] - 3s 4ms/step - loss: 3.1603 - accuracy: 0.2903  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 2/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.5875 - accuracy: 0.8011  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 3/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.3313 - accuracy: 0.8876  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 4/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.2551 - accuracy: 0.9186  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 5/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1970 - accuracy: 0.9409  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 6/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1700 - accuracy: 0.9487  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 7/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1772 - accuracy: 0.9467  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 8/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1934 - accuracy: 0.9438  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 9/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1232 - accuracy: 0.9663  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 10/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1761 - accuracy: 0.9479  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 11/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1337 - accuracy: 0.9632  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 12/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1270 - accuracy: 0.9634  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.  
Epoch 13/50  
445/445 [==============================] - 2s 4ms/step - loss: 0.1580 - accuracy: 0.9557  
WARNING:tensorflow:Can save best model only with val\_loss available, skipping.

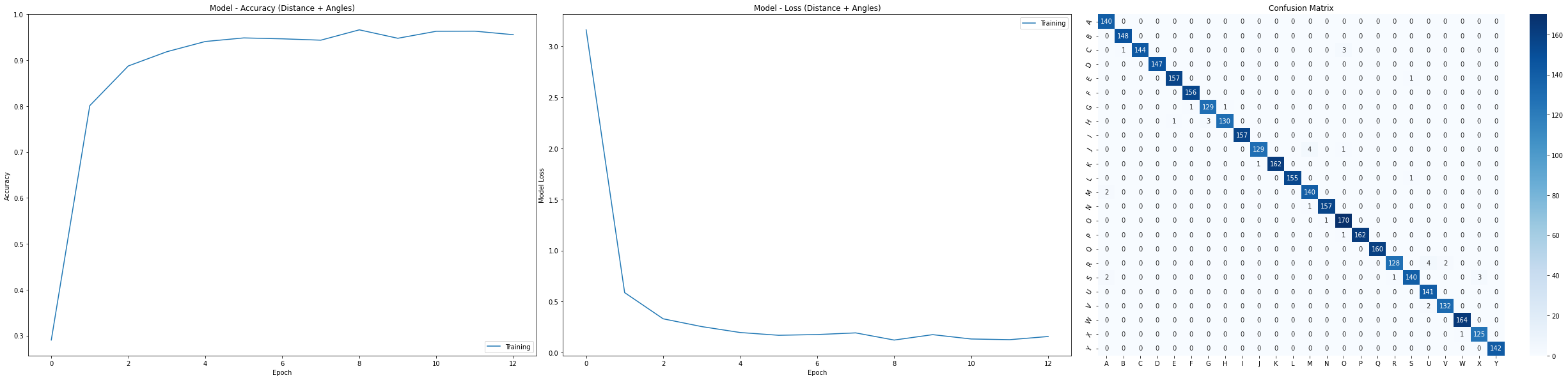
[124]

**from** tensorflow.keras.models **import** load\_model  
  
model\_angle\_and\_distance = load\_model(**'./model\_saves/angles\_distance'**)

[136]

angle\_and\_distance\_results = evaluate\_model(history, X\_test, y\_test, model\_angle\_and\_distance, **"Distance + Angles"**, actions)

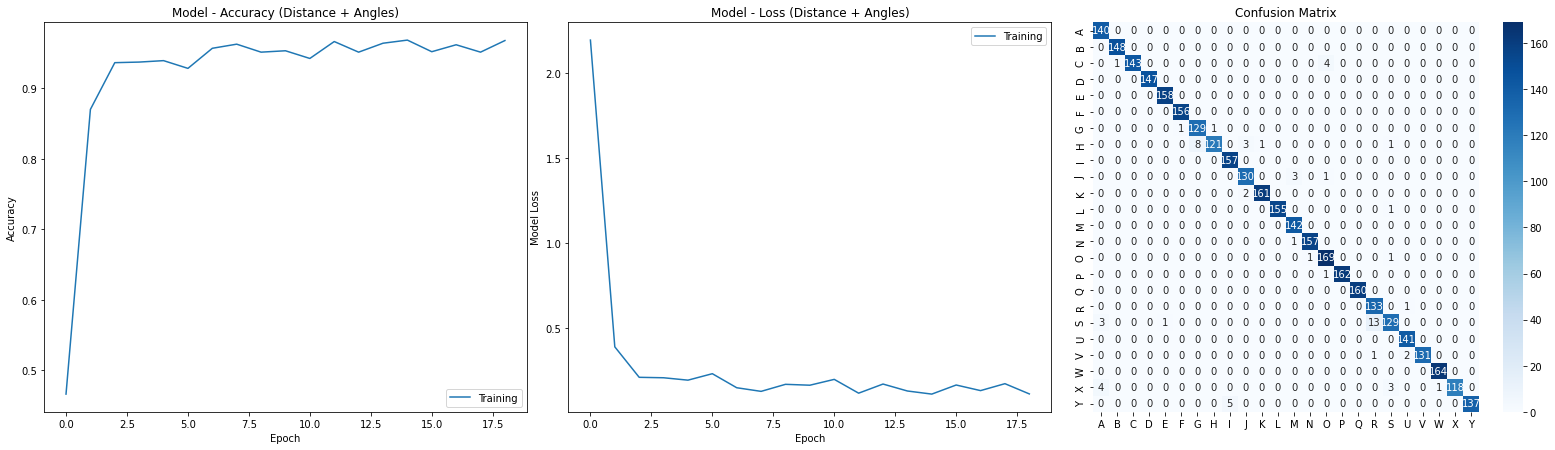
Accuracy: 98.93%   
  
 precision recall f1-score support  
  
 A 0.97 1.00 0.99 140  
 B 0.99 1.00 1.00 148  
 C 1.00 0.97 0.99 148  
 D 1.00 1.00 1.00 147  
 E 0.99 0.99 0.99 158  
 F 0.99 1.00 1.00 156  
 G 0.98 0.98 0.98 131  
 H 0.99 0.97 0.98 134  
 I 1.00 1.00 1.00 157  
 J 0.99 0.96 0.98 134  
 K 1.00 0.99 1.00 163  
 L 1.00 0.99 1.00 156  
 M 0.97 0.99 0.98 142  
 N 0.99 0.99 0.99 158  
 O 0.97 0.99 0.98 171  
 P 1.00 0.99 1.00 163  
 Q 1.00 1.00 1.00 160  
 R 0.99 0.96 0.97 134  
 S 0.99 0.96 0.97 146  
 U 0.96 1.00 0.98 141  
 V 0.99 0.99 0.99 134  
 W 0.99 1.00 1.00 164  
 X 0.98 0.99 0.98 126  
 Y 1.00 1.00 1.00 142  
  
 accuracy 0.99 3553  
 macro avg 0.99 0.99 0.99 3553  
weighted avg 0.99 0.99 0.99 3553



[46]

angle\_and\_distance\_results = evaluate\_model(history, X\_test, y\_test, model\_angle\_and\_distance, **"Distance + Angles"**, actions)

Accuracy: 98.17%   
  
 precision recall f1-score support  
  
 A 0.95 1.00 0.98 140  
 B 0.99 1.00 1.00 148  
 C 1.00 0.97 0.98 148  
 D 1.00 1.00 1.00 147  
 E 0.99 1.00 1.00 158  
 F 0.99 1.00 1.00 156  
 G 0.94 0.98 0.96 131  
 H 0.99 0.90 0.95 134  
 I 0.97 1.00 0.98 157  
 J 0.96 0.97 0.97 134  
 K 0.99 0.99 0.99 163  
 L 1.00 0.99 1.00 156  
 M 0.97 1.00 0.99 142  
 N 0.99 0.99 0.99 158  
 O 0.97 0.99 0.98 171  
 P 1.00 0.99 1.00 163  
 Q 1.00 1.00 1.00 160  
 R 0.90 0.99 0.95 134  
 S 0.96 0.88 0.92 146  
 U 0.98 1.00 0.99 141  
 V 1.00 0.98 0.99 134  
 W 0.99 1.00 1.00 164  
 X 1.00 0.94 0.97 126  
 Y 1.00 0.96 0.98 142  
  
 accuracy 0.98 3553  
 macro avg 0.98 0.98 0.98 3553  
weighted avg 0.98 0.98 0.98 3553



[77]

model\_two\_angles.save(**'./model\_saves/angles\_distance'**)

INFO:tensorflow:Assets written to: ./model\_saves/angles\_distance\assets

[47]

*# with open('data/results.pkl', 'rb') as f:*  
*#     list\_of\_metrics = pickle.load(f)*  
  
list\_of\_metrics[**'angle+distance'**] = angle\_and\_distance\_results  
  
**with** open(**'data/results\_new.pkl'**, **'wb'**) **as** f:  
     pickle.dump(list\_of\_metrics, f)